BAYESIAN OPTIMIZATION WITH PARALLEL FUNCTION EVALUATIONS AND MULTIPLE INFORMATION SOURCES: METHODOLOGY WITH APPLICATIONS IN BIOCHEMISTRY, AEROSPACE ENGINEERING, AND MACHINE LEARNING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Jialei Wang

January 2017

BAYESIAN OPTIMIZATION WITH PARALLEL FUNCTION EVALUATIONS AND MULTIPLE INFORMATION SOURCES: METHODOLOGY WITH APPLICATIONS IN BIOCHEMISTRY, AEROSPACE ENGINEERING, AND MACHINE LEARNING

Jialei Wang, Ph.D.

Cornell University 2017

Bayesian optimization, a framework for global optimization of expensive-to-evaluate functions, has recently gained popularity in machine learning and global optimization because it can find good feasible points with few function evaluations. In this dissertation, we present novel Bayesian optimization algorithms for problems with parallel function evaluations and multiple information sources, for use in machine learning, biochemistry, and aerospace engineering applications.

First, we present a novel algorithm that extends expected improvement, a widely-used Bayesian optimization algorithm that evaluates one point at a time, to settings with parallel function evaluations. This algorithm is based on a new efficient solution method for finding the Bayes-optimal set of points to evaluate next in the context of parallel Bayesian optimization. The author implemented this algorithm in an open source software package co-developed with engineers at Yelp, which was used by Yelp and Netflix for automatic tuning of hyperparameters in machine learning algorithms, and for choosing parameters in online content delivery systems based on evaluations in A/B tests on live traffic.

Second, we present a novel parallel Bayesian optimization algorithm with a worst-case approximation guarantee applied to peptide optimization in biochemistry, where we face a large collection of peptides with unknown fitness prior to

experimentation, and our goal is to identify peptides with a high score using a small number of experiments. High scoring peptides can be used for biolabeling, targeted drug delivery, and self-assembly of metamaterials. This problem has two novelties: first, unlike traditional Bayesian optimization, where the objective function has a continuous domain and real-valued output well-modeled by a Gaussian Process, this problem has a discrete domain, and involves binary output not well-modeled by a Gaussian process; second, it uses hundreds of parallel function evaluations, which is a level of parallelism too large to be approached with other previously-proposed parallel Bayesian optimization methods.

Third, we present a novel Bayesian optimization algorithm for problems in which there are multiple methods or "information sources" for evaluating the objective function, each with its own bias, noise and cost of evaluation. For example, in aerospace engineering, to evaluate an aircraft wing design, different computational models may simulate performance. Our algorithm explores the correlation and model discrepancy of each information source, and optimally chooses the information source to evaluate next and the point at which to evaluate it. We describe how this algorithm can be used in general multi-information source optimization problems, and also how a related algorithm can be used in "warm start" problems, where we have results from previous optimizations of closely related objective functions, and we wish to leverage these results to more quickly optimize a new objective function.

BIOGRAPHICAL SKETCH

Jialei Wang was born and raised in Qidong, in JiangSu Province, China, a small but beautiful city by the East China Sea and the Yangtze River. Before he went to college, he spent most of his time in his hometown. After graduating from the high school, he wanted to start a new venture away from the place that he was too familiar with, and in 2007, he went to Nanyang Technological University in Singapore to study Physics. He then transferred to the University of Illinois at Urbana-Champaign, from which he received a B.Sc. in Physics with the highest honors in 2011. In the same year, he joined the Ph.D. program in Applied Physics at Cornell University. In the summer of 2012, he began to work with Professor Peter I. Frazier from the department of Operations Research, and later on, he realized that Operations Research was the right field for him. Professor Peter I. Frazier also became his adviser, and supervised the work in this dissertation.

To my wife, Ziqi.

It is your constant love and support,

that made this work possible.

ACKNOWLEDGEMENTS

I am grateful to many people for their help in completing my Ph.D.

First and foremost, I would like to thank my adviser, Professor Peter I. Frazier, for constantly helping me to grow both as a researcher and a person. His ability to apply Operations Research to problems from very different fields and collaborate with people from very different background amazes me and sets a great example for me, that continues to encourage me to apply Operations Research to a broader area of problems going forward. His untiring availability for guiding my research and offering advice in cracking problems along the road, made it possible for completing this dissertation work. I also thank Peter, for his generosity in funding my entire Ph.D. research, including attending research conferences, and providing computational resources for conducting numerical experiments.

I am also grateful to my family. I would like to thank my wife, Ziqi, for her constantly caring and support. Failures and stress frequently occur in the journey of doing research, and while I was being upset, it was always Ziqi's encouragement that brought me back on track, and motivated me until I saw the dawn of success. I also thank my parents, for raising me with great care, offering me the best education opportunities, and setting me a great example through the way they live their lives.

Finally, I would like to thank everyone at Cornell, in particular within my department of Operations Research & Information Engineering, for the marvelous atmosphere of kindness and intelligence they have created. I am grateful to my minor advisers, Professor Thorsten Joachims and Professor Paulette Clancy, for their valuable inputs to this dissertation work. I would also like to thank my research collaborators, for their hard working and great teamwork on the challenges we have taken on.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis considers derivative-free global optimization of expensive functions, in which (1) our objective function is time-consuming to evaluate, limiting the number of function evaluations we can perform; (2) evaluating the objective function provides only the value of the objective, and not the gradient or Hessian; (3) we seek a global, rather than a local, optimum. Such problems typically arise when the objective function is evaluated by running a complex computer code (see, e.g., [76]), but also arises when the objective function can only be evaluated by performing a laboratory experiment, or building a prototype system to be evaluated in the real world.

Bayesian optimization (BO) methods constitute one class of methods attempting to solve such problems, where they use machine learning to build a predictive model for the unknown objective function, and then use decision theory to suggest which point(s) in the function's domain would be the most valuable to evaluate next. Bayesian optimization was pioneered by [53], with early work through the 1970s and 1980s being pursued in [65] and [63]. Development in the 1990s was marked by the popularization of Bayesian optimization by Jones, Schonlau, and Welch, who, building on previous work by Mockus, introduced the Efficient Global Optimization (EGO) method [42]. This method became very popular and well-known in engineering, where it has been adopted for design applications involving time-consuming computer experiments. In the 2000s, development of Bayesian optimization continued in statistics and engineering, and the 2010s have seen additional development from the machine learning community, where Bayesian optimization is used for tuning hyperparameters of computationally expensive machine

learning models [86]. Other introductions to Bayesian optimization may be found in the tutorial article [8] and textbooks [18, 77], and an overview of the history of the field may be found in [78].

In this chapter, we first collect a few problems arising from engineering and machine learning that are suitable for Bayesian optimization in Section 1.1. We then present the precise problem formulation considered by Bayesian optimization in Section 1.2, and discuss the predictive technique used by Bayesian optimization, called Gaussian Process (GP) regression, in Section 1.2.1, and the methods of suggesting the point(s) to evaluate next in Section 1.2.2. In Section 1.3, we discuss potential usage of Bayesian optimization to the problems in cheminformatics, and additional work need to be done. Finally, we provide an overview for the rest of the thesis in Section 1.4.

1.1 Examples of Bayesian Optimization Problems

Bayesian optimization problems are incredibly common in application. We collect a few examples, arising from engineering and machine learning. Some of these examples will be considered more fully later, and others are included to underscore the broad scope encompassed by the class of Bayesian optimization problems.

- **Nano-materials design:** We would like to choose design variables to some nano-materials fabrication process to maximize some measure of the performance of the design. For example, in the fabrication of nanocrystalline silicon for high-performance / low-power transistor circuit technology, the design variables could be the choice of flow rate, pressure, temperature, radiofrequency power, etc., and the performance measure would be the mobility

of the charge carrier in the transistor channel. Evaluating a design requires making the material and testing it, which takes time, material cost, and manpower. See [45].

- **Oil exploration:** We would like to discover the best place at which to drill a commercial oil well by drilling a sequence of exploratory test wells. We would like to find a good location with as few exploratory as possible in the discovery process. See [1].
- **Drug discovery:** We would like to search among chemically similar derivatives of a molecule to find the one that best treats a given disease. The number of these molecules is often too large to test all of them in experiments, therefore, we test the molecules adaptively, and at each point in time, we decide which molecule to test, and then collect information about its effectiveness. See [66].
- **Peptide optimization:** We would like to iteratively discover and refine functional peptides with a desired physical / biochemical property, which would support innovations in medicine, biochemistry, and materials science. Few peptides may have the desired property, making the search difficult, because the size of the peptide library to be searched grows exponentially with the maximum peptide length considered. See [24].
- **Tuning hyperparameters of machine learning model:** For some machine learning algorithms, e.g., deep neural networks, using high-quality hyperparameters instead of low-quality ones is the difference between state-of-the-art predictive performance and being essentially useless. Typical approaches to tuning hyperparameters include hand tuning, by experts and brute-force search. However, as the number of parameters grow, these approaches quickly become infeasible. To overcome this challenge, Bayesian

optimization methods can be used to automate hyperparameter tuning. See [86].

1.2 The Bayesian Optimization Approach

Bayesian optimization considers optimizing a nonlinear function $f(\boldsymbol{x})$ over a compact set $\mathbb{A} \subset \mathbb{R}^d$, formulated concisely as follows:

$$\max_{\boldsymbol{x} \in \mathbb{A} \subset \mathbb{R}^d} f(\boldsymbol{x}). \tag{1.1}$$

In many realistic problems of this kind, we do not know a specific analytic form for the objective function, and can only hope to obtain an estimate of the objective function through simulations or conducting laboratory experiments. Derivative information about the objective functions is not available either. Moreover, sampling from $f(\boldsymbol{x})$ is usually an expensive process, for example, conducting an experiment on a new drug design takes days for synthesizing and testing the molecule, computing performance metric of a complex machine learning model corresponding to one hyperparameter setting requires hours of computing time, etc. We also assume the problem is non-convex.

Bayesian optimization is particularly suitable for these problems, when the objective function does not have an explicit analytic form, derivative information is not available, and function evaluation is costly. The merit of the Bayesian optimization approach stems from two aspects: first, Bayesian optimization can incorporate domain expertise about the problem in the form of a Bayesian prior belief; second, the decision-theoretic approach to sampling ensures that each sample is chosen to improve our solution to the optimization problem as much as possible, reducing the number of function evaluations required to find the optimum. We

will revisit both aspects in Section 1.2.1 and Section 1.2.2.

1.2.1 The Predictive Model: Gaussian Process Regression

Since the objective function is assumed unknown, in Bayesian optimization, we first build a predictive model on the objective function using Bayesian statistics. The name *Bayesian* comes from the famous "Bayes' theorem", which states that the *posterior distribution* of a random quantity θ given observational data $\mathcal{D}$, is proportional to the *likelihood* of $\mathcal{D}$ given θ multiplied by the *prior* distribution of θ:

$$\mathbb{P}(\theta \mid \mathcal{D}) \propto \mathbb{P}(\mathcal{D} \mid \theta)\mathbb{P}(\theta). \tag{1.2}$$

Bayesian models have a few properties that make them well-suited for Bayesian optimization. First, Bayesian models provide a probability distribution over the quantity being predicted, called the *posterior*, and not just a point estimator. Having a probability distribution is crucial in Bayesian optimization, as will become clear in Section 1.2.2. Second, Bayesian models make inference about the model parameters based on information obtained from both the observational data and the prespecified prior distribution. Thus one can offer more information than the data itself to the model, by encoding prior knowledge about the model parameters in the prior distribution. Since Bayesian optimization problems typically begin with a small dataset due to the expense of doing function evaluations, Bayesian models' ability to pool additional information from domain experts helps alleviate the "cold start" issue at the beginning of the optimization routine. While optimization methods without Bayesian models may leverage the domain experts' knowledge by choosing a "good" starting point, it is difficult to more fully encode domain expertise into these methods. In addition, when new observations become

available, the previous posterior distribution can be used as a prior, and inference logically follow from Bayes' theorem, allowing convenient updates to Bayesian models, without needing to fully re-train the model on all previous data.

Among the wide variety of Bayesian statistical methods, Gaussian Process (GP) regression is a popular choice in Bayesian optimization. A Gaussian Process is a probability distribution over functions. Under the Gaussian Process, the marginal probability distribution of the value of the function at any single point is a normal distribution. The joint distribution of the values of the function at any collection of points is a multivariate normal distribution.

In Gaussian process regression, we use a Gaussian Process as our prior probability distribution over the unknown objective function. We first place a Gaussian Process prior over the unknown objective function f, and when data become available, we use Bayes' theorem to update the posterior, following (1.2), with $\theta = f$.

The detailed specification of the prior, and the updating scheme, is as follows: a Gaussian Process is completely specified by its mean function, $\mu(\boldsymbol{x}) : \mathbb{A} \mapsto \mathbb{R}$, and covariance function, $k(\boldsymbol{x}, \boldsymbol{x}') : \mathbb{A} \times \mathbb{A} \mapsto \mathbb{R}$. When we place a Gaussian Process prior over the function f, we write it as

$$f \sim \mathcal{GP}(\mu, k). \tag{1.3}$$

For a collection of points $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_q)$, the prior of f evaluated at each of the points in $\boldsymbol{X}$ is a multivariate Gaussian distribution

$$f(\boldsymbol{X}) \sim \mathcal{N}\left(\boldsymbol{\mu}^{(0)}, \boldsymbol{\Sigma}^{(0)}\right), \tag{1.4}$$

where $\boldsymbol{\mu}_i^{(0)} = \mu(\boldsymbol{x}_i)$, and $\boldsymbol{\Sigma}_{ij}^{(0)} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i, j \in \{1, \ldots, q\}$. Therefore, $\boldsymbol{\mu}^{(0)}$ is a $q \times 1$ column vector, and $\boldsymbol{\Sigma}^{(0)}$ is a $q \times q$ square matrix. Note that the subscripts

of the notations identify which point in the given point collection, while the superscript denotes the number of samples observed for updating the posterior. We use the superscript 0 to denote a prior, because at the moment we do not observe any sample. Suppose we have evaluated the objective function at the points $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$, with the corresponding values $y^{(1)}, \ldots, y^{(n)}$, and we use $\boldsymbol{x}^{(1:n)}$ and $y^{(1:n)}$ to simplify the notations, then we can compute the posterior of f over $\boldsymbol{X}$ using Bayes' theorem, written as

$$f(\boldsymbol{X}) \mid \boldsymbol{X}, \boldsymbol{x}^{(1:n)}, y^{(1:n)} \sim \mathcal{N}\left(\boldsymbol{\mu}^{(n)}, \boldsymbol{\Sigma}^{(n)}\right). \tag{1.5}$$

If the function evaluation is noise-free, meaning $y^{(i)} = f(\boldsymbol{x}^{(i)}), i = 1, \ldots, n$, the formula for $\boldsymbol{\mu}^{(n)}$ and $\boldsymbol{\Sigma}^{(n)}$ is

$$\begin{aligned}
\boldsymbol{\mu}^{(n)} &= \boldsymbol{\mu}^{(0)} + K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right) K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}\right)^{-1} \left(y^{(1:n)} - \mu^{(1:n)}\right), \\
\boldsymbol{\Sigma}^{(n)} &= K\left(\boldsymbol{X}, \boldsymbol{X}\right) - K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right) K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}\right)^{-1} K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{X}\right),
\end{aligned} \tag{1.6}$$

where $\mu^{(1:n)}$ is a n-dimensional vector with each component $\mu^{(i)} = \mu(\boldsymbol{x}^{(i)}), i = 1, \ldots, n$; $K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right)$ is a $q \times n$ matrix with $K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right)_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}^{(j)})$, and similarly for $K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{X}\right)$, $K\left(\boldsymbol{X}, \boldsymbol{X}\right)$ and $K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}\right)$.

When function evaluations are noisy, and we assume additive independent identically distributed Gaussian noise with variance σ^2, then we can write $\boldsymbol{\mu}^{(n)}$ and $\boldsymbol{\Sigma}^{(n)}$ as

$$\begin{aligned}
\boldsymbol{\mu}^{(n)} &= \boldsymbol{\mu}^{(0)} + K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right) \left[K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}\right) + \sigma^2 I\right]^{-1} \left(y^{(1:n)} - \mu^{(1:n)}\right), \\
\boldsymbol{\Sigma}^{(n)} &= K\left(\boldsymbol{X}, \boldsymbol{X}\right) - K\left(\boldsymbol{X}, \boldsymbol{x}^{(1:n)}\right) \left[K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}\right) + \sigma^2 I\right]^{-1} K\left(\boldsymbol{x}^{(1:n)}, \boldsymbol{X}\right).
\end{aligned} \tag{1.7}$$

[74, Sect. 2.2] provides the details of the derivation.

Figure 1.1 shows the output from Gaussian process regression on a one-dimensional function. In the figure, circles show points $(x^{(i)}, f(x^{(i)}))$, the solid

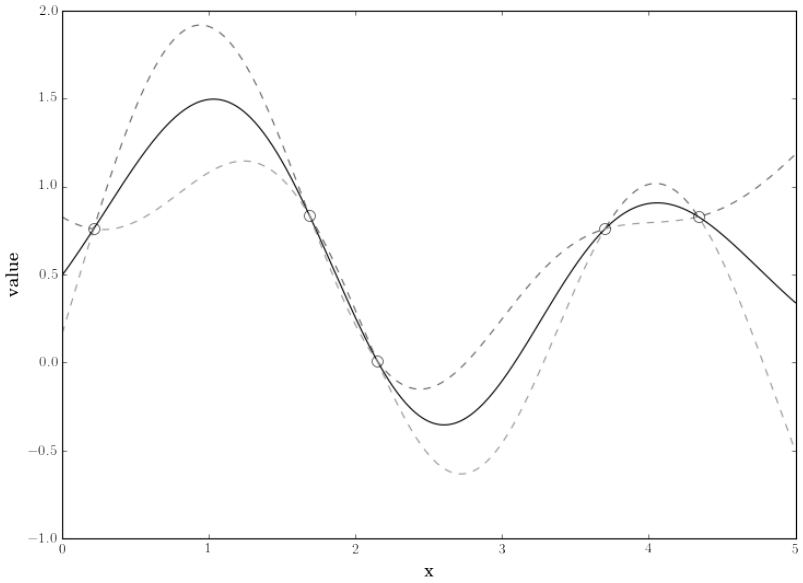


Figure 1.1: Illustration of Gaussian process regression with noise-free evaluations. The circles show previously evaluated points, $(x^{(i)}, f(x^{(i)}))$. The solid line shows the posterior mean, $\mu^{(n)}(x)$ as a function of x, which is an estimate $f(x)$, and the dashed lines show a Bayesian credible interval for each $f(x)$, calculated as $\mu^{(n)}(x) \pm 1.96\sigma^{(n)}(x)$. Although this example shows f taking a scalar input, Gaussian process regression can be used for functions with vector inputs.

line shows $\mu^{(n)}(x)$ as a function of x, and the dashed lines are positioned at $\mu^{(n)}(x) \pm 1.96\sigma^{(n)}(x)$, forming a 95% Bayesian credible interval for $f(x)$, i.e., an interval in which $f(x)$ lies with posterior probability 95%. (A credible interval is the Bayesian version of a frequentist confidence interval.) Because observations are noise-free, the posterior mean $\mu^{(n)}(x)$ interpolates the observations $f(x)$.

[21] offers a comprehensive review of Gaussian process regression, including the choice of mean function and covariance function, inference with noisy observations, and hyperparameters estimation.

1.2.2 Choosing Where to Sample: Acquisition Functions for Bayesian Optimization

The second crucial piece of Bayesian optimization is making good decisions about where to direct future sampling. Bayesian optimization methods address this by using a measure of the value of the information that would be gained by sampling at a point, commonly known as an "acquisition function". Bayesian optimization methods then choose the point to sample that maximizes this acquisition function. A number of different acquisition functions have been proposed, and here we describe three in detail; *probability of improvement* [53], *expected improvement* [42, 64], and the *knowledge gradient* [19, 79]. A more comprehensive review of these acquisition functions may be found in [8, 21]. Other acquisition functions not discussed here include entropy search [92], and composite measures involving the mean and the standard deviation of the posterior [38].

Acquisition Function: Probability of Improvement

Considering the setting of noise-free function evaluations, the early work of [53] suggested maximizing the probability of improvement over the best sampled value observed so far, written as

$$\texttt{PI}(\boldsymbol{x}) = \mathbb{P}\left(f(\boldsymbol{x}) \geq f_n^* + \epsilon\right), \tag{1.8}$$

where $f_n^* = \max_{m \leq n} f(\boldsymbol{x}^{(m)})$, is the best sampled value at nth iteration, and ϵ is a positive constant that controls how much improvement over the current best sampled value is desired. Recall in (1.5) that if we have not observed $f(\boldsymbol{x})$ yet, $f(\boldsymbol{x})$ is a random variable that follows a normal distribution with mean $\mu^{(n)}(\boldsymbol{x}) = \boldsymbol{\mu}_1^{(n)}$, and standard deviation $\sigma^{(n)}(\boldsymbol{x}) = \left(\boldsymbol{\Sigma}_{1,1}^{(n)}\right)^{\frac{1}{2}}$, when we set $\boldsymbol{X} = \{\boldsymbol{x}\}$. The superscript

n here means the distribution is a posterior after observing n data points. Then we can write (1.8) as

$$\texttt{PI}(\boldsymbol{x}) = \Phi\left(\frac{\mu^{(n)}(\boldsymbol{x}) - f_n^* - \epsilon}{\sigma^{(n)}(\boldsymbol{x})}\right), \tag{1.9}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function.

The choice of ϵ is left to users, although [53] suggested that in general ϵ should start with a bigger value at the beginning of the sampling procedure, to ensure enough exploration of the function domain, and gradually decrease to zero toward the end of sampling to engage more effort in exploitation. Several works have studied the empirical impact of choices of ϵ [90, 41, 58].

Figure 1.2 shows the behavior of probability of improvement for a problem with a one-dimensional input space, and the effect of varying ϵ. We can see that probability of improvement is largest at locations near the current best sampled value. If we did not introduce ϵ, i.e., we set $\epsilon = 0$, the probability of improvement algorithm would become identical to pure exploitation, because the point with highest probability of being greater than the current best sampled value would simply be the best previously sampled point, as shown in the figure. To add exploration to the algorithm, we must increase ϵ. When ϵ is large, Figure 1.2 confirms that as we set ϵ larger, the algorithm favors points that have large potential gain over the current best sampled value.

Acquisition Function: Expected Improvement

A more satisfying alternative acquisition function would not only consider the probability of improvement, but also the magnitude of the improvement. [42] proposed such an alternative, called expected improvement. As its name suggests,

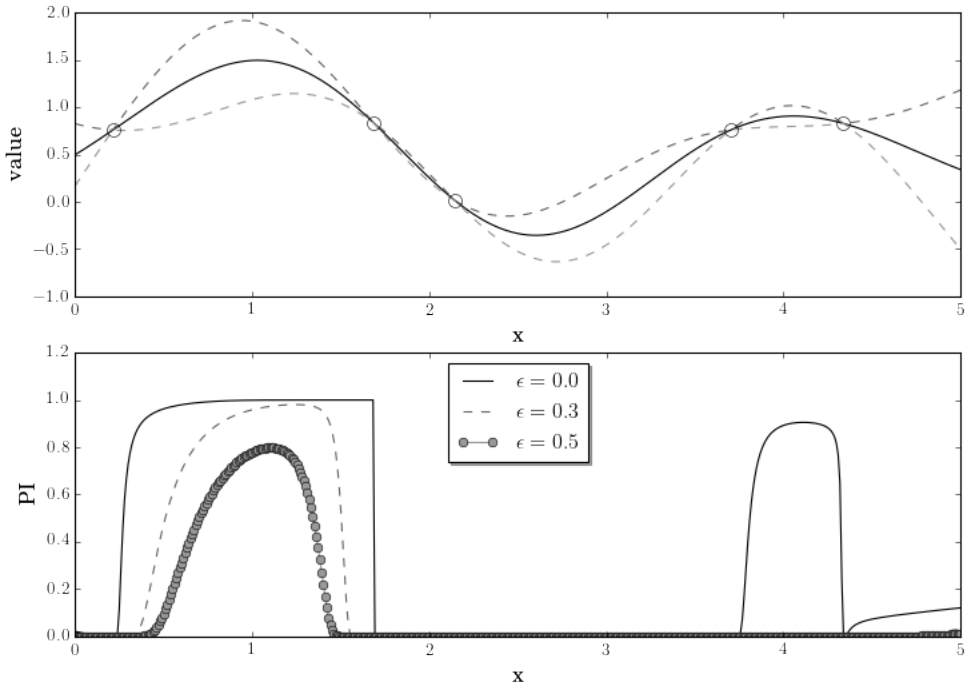


Figure 1.2: Upper panel shows the posterior distribution in a problem with no noise and a one-dimensional input space, where the circles are previously sampled points, the solid line is the posterior mean $\mu^{(n)}(x)$, and the dashed lines are at $\mu^{(n)}(x) \pm 2\sigma^{(n)}(x)$. Lower panel shows the probability of improvement $\texttt{PI}(x)$ computed from this posterior distribution. Three different ϵ values were used in computing probability of improvement to show the effect of ϵ in controlling the exploration v.s. exploitation behavior.

the formulation is

$$\texttt{EI}(\boldsymbol{x}) = E_n\left[(f(\boldsymbol{x}) - f_n^*)^+\right], \tag{1.10}$$

where $E_n[\cdot]$ is the conditional expectation given previous n evaluations. The expectation in (1.10) can be written more explicitly, in terms of the normal cumulative distribution function $\Phi(\cdot)$, and the normal probability density function $\varphi(\cdot)$:

$$\texttt{EI}(\boldsymbol{x}) = \left(\mu^{(n)}(\boldsymbol{x}) - f_n^*\right)\Phi\left(\frac{\mu^{(n)}(\boldsymbol{x}) - f_n^*}{\sigma^{(n)}(\boldsymbol{x})}\right) + \sigma^{(n)}(\boldsymbol{x})\varphi\left(\frac{\mu^{(n)}(\boldsymbol{x}) - f_n^*}{\sigma^{(n)}(\boldsymbol{x})}\right). \tag{1.11}$$

The advantage of this formulation compared with the probability of improvement is that, without a user defined controlling variable ϵ, the expected improve-

ment balances trade-off between exploration and exploitation automatically. In fact, the expected improvement favors points that, on the one hand, have a large predicted value, while on the other hand, have a significant amount of uncertainty to allow room for improvement.

We illustrate this behavior in Figure 1.3, which plots the expected improvement for a problem with a one-dimensional input space. We can see from this plot that the expected improvement is largest at locations where both the posterior mean $\mu^{(n)}(x)$ is large, and also the posterior standard deviation $\sigma^{(n)}(x)$ is large. This is reasonable because those points that are most likely to provide large gains are those points that have a high predicted value, but that also have significant uncertainty. Indeed, at points where we have already observed, and thus have no uncertainty, the expected improvement is 0. This is consistent with the idea that, in a problem without noise, there is no value to repeating an evaluation that has already been performed.

In many applications, e.g., those involving physical experiments or stochastic simulations, measurements are noisy. The formulation (1.10) is not applicable in this case, because (1) f_n^* is not well-defined under noisy observations, and (2) it does not account for the prediction uncertainty at the current best point. To alleviate these difficulties, alternative formulations of expected improvement were proposed in literature. For example, [39] proposed *augmented expected improvement*, which changed the original formulation of expected improvement to adapt to the noisy setting: (1) it used the Gaussian Process updating equations for noisy measurements, given in (1.7); (2) it defined the current "effective best solution" as $\max_{m \leq n} \left(\mu^{(n)}(\boldsymbol{x}^{(m)}) - c \cdot \sigma^{(n)}(\boldsymbol{x}^{(m)}) \right)$ to replace the original definition of the current best solution, where c is a constant that can reflect the users' degree

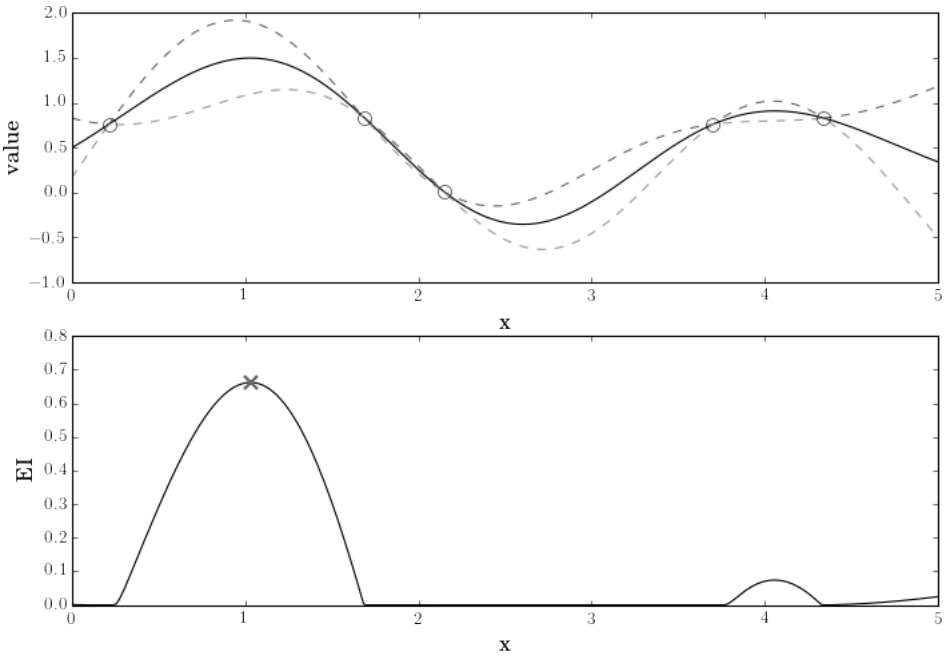


Figure 1.3: Upper panel shows the posterior distribution in a problem with no noise and a one-dimensional input space, where the circles are previously measured points, the solid line is the posterior mean $\mu^{(n)}(x)$, and the dashed lines are at $\mu^{(n)}(x) \pm 2\sigma^{(n)}(x)$. Lower panel shows the expected improvement $\texttt{EI}(x)$ computed from this posterior distribution. An "x" is marked at the point with the largest expected improvement, which is where we would evaluate next.

of risk aversion; (3) it discounted the original expected improvement by a factor of $(1 - \frac{\sigma}{\sqrt{(\sigma^{(n)}(\boldsymbol{x}))^2+\sigma^2}})$, to account for the diminishing return of additional replicates as the prediction becomes more accurate. In addition to this work, there are other approaches in extending expected improvement to the noisy setting proposed in literature; refer to [40, 68] for detail.

Acquisition Function: Knowledge Gradient

Knowledge gradient [20, 79] fully accounts for the introduction of noise, and makes it possible to search over a class of solutions broader than just those that have been

previously evaluated when recommending the final solution.

We first introduce a set A_n, which is the set of points from which we would choose final solution, if we were asked to recommend a final solution at time n, based on $\boldsymbol{x}^{(1:n)}$, $y^{(1:n)}$. For tractability, we suppose A_n is finite. For example, if A is finite, as it often is in discrete optimization via simulation problems, we could take $A_n = A$, allowing the whole space of feasible solutions. This choice was considered in [20]. Alternatively, one could take $A_n = \boldsymbol{x}^{(1:n)}$, stating that one is willing to consider only those points that have been previously evaluated. This choice is consistent with the expected improvement algorithm. Indeed, we will see that when one makes this choice, and measurements are free from noise, then the knowledge gradient algorithm is identical to the expected improvement algorithm. Thus, the knowledge gradient algorithm generalizes the expected improvement algorithm.

If we were to stop sampling at time n, then the expected value of a point $\boldsymbol{x} \in A_n$ based on the information available would be $E_n[f(\boldsymbol{x})] = \mu^{(n)}(\boldsymbol{x})$. In the special case when evaluations are free from noise, this is equal to $f(\boldsymbol{x})$, but when there is noise, these two quantities may differ. If we needed to report a final solution, we would then choose the point in A_n for which this quantity is the largest, i.e., we would choose $\operatorname{argmax}_{\boldsymbol{x}\in A_n} \mu^{(n)}(\boldsymbol{x})$. Moreover, the expected value of this solution would be

$$\mu_n^* = \max_{\boldsymbol{x}\in A_n} \mu^{(n)}(\boldsymbol{x}).$$

If evaluations are free from noise and $A_n = \{\boldsymbol{x}^{(1:n)}\}$, then μ_n^* is equal to f_n^*, but in general these quantities may differ.

If we take one additional sample, then the expected value of the solution we

would report based on this additional information is

$$\mu^*_{n+1} = \max_{x \in A_{n+1}} \mu^{(n+1)}(\boldsymbol{x}),$$

where as before, A_{n+1} is some finite set of points we would be willing to consider when choosing a final solution. Observe in this expression that $\mu^{(n+1)}(\boldsymbol{x})$ is not necessarily the same as $\mu^{(n)}(\boldsymbol{x})$, even for points $\boldsymbol{x} \in \{\boldsymbol{x}^{(1:n)}\}$ that we had previously evaluated, but that $\mu^{(n+1)}(\boldsymbol{x})$ can be computed from the history of observations $\boldsymbol{x}^{(1:n+1)}$, $y^{(1:n+1)}$.

The improvement in our expected solution value is then the difference between these two quantities, $\mu^*_{n+1} - \mu^*_n$. This improvement is random at time n, even fixing $\boldsymbol{x}^{(n+1)}$, through its dependence on $y^{(n+1)}$, but we can take its expectation. The resulting quantity is called the knowledge gradient (KG) factor, and is written,

$$\texttt{KG}_n(\boldsymbol{x}) = E_n \left[\mu^*_{n+1} - \mu^*_n \mid \boldsymbol{x}^{(n+1)} = \boldsymbol{x} \right]. \tag{1.12}$$

Calculating this expectation is more involved than calculating the expected improvement, but nevertheless can also be done analytically in terms of the normal pdf and normal cdf. This is described in more detail in [20].

The knowledge gradient algorithm is then the one that chooses the point to sample next that maximizes the KG factor,

$$\operatorname*{argmax}_{\boldsymbol{x}} \texttt{KG}_n(\boldsymbol{x}).$$

The KG factor for a one-dimensional optimization problem with noise is pictured in Figure 1.4. We see a similar trade-off between exploration and exploitation, where the KG factor favors measuring points with a large $\mu^{(n)}(\boldsymbol{x})$ and a large $\sigma^{(n)}(\boldsymbol{x})$. We also see local minima of the KG factor at points where we previously

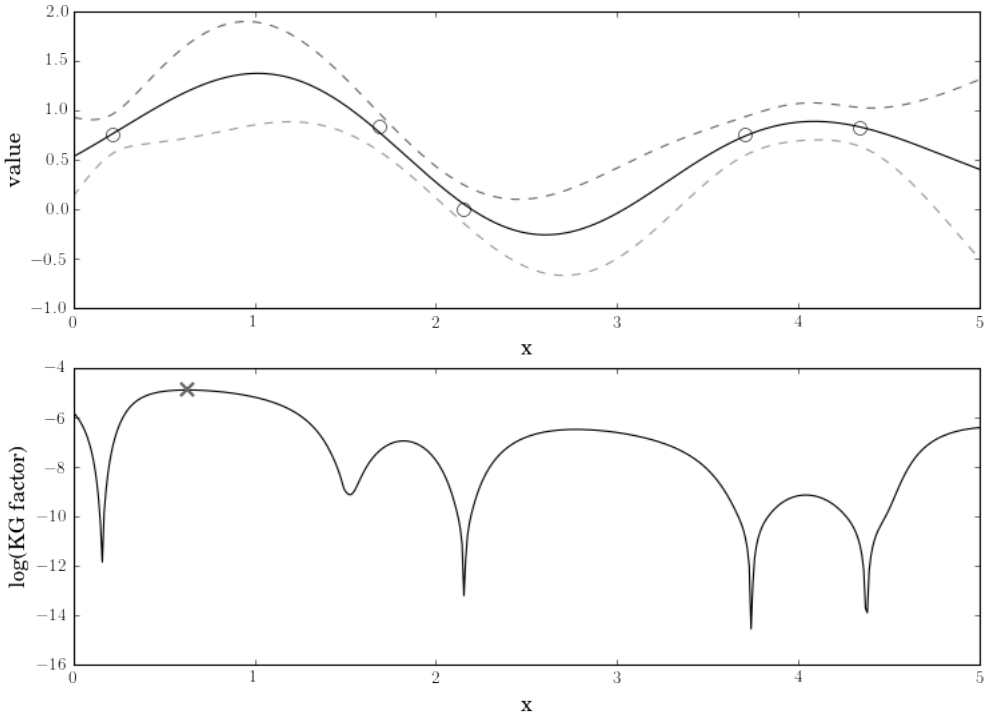


Figure 1.4: Upper panel shows the posterior distribution in a problem with independent normal homoscedastic noise and a one-dimensional input space, where the circles are previously measured points, the solid line is the posterior mean $\mu^{(n)}(x)$, and the dashed lines are at $\mu^{(n)}(x) \pm 2\sigma^{(n)}(x)$. Lower panel shows the natural logarithm of the knowledge gradient factor $\texttt{KG}(x)$ computed from this posterior distribution, where $A_n = A_{n+1}$ are the discrete grid of 500 points in the range $[0, 5]$. An "x" is marked at the point with the largest KG factor, which is where the KG algorithm would evaluate next.

evaluated, just as with the expected improvement, but because there is noise in our samples, the value at these points is not 0 — indeed, when there is noise, it may be useful to sample repeatedly at a point.

Recall that the KG factor depends on the choice of the sets A_n and A_{n+1}. For noise-free problems, if we set $A_{n+1} = \{\boldsymbol{x}^{(1:n+1)}\}$ and $A_n = \{\boldsymbol{x}^{(1:n)}\}$, we will see that the KG factor recovers expected improvement. Typically, to achieve a better result, we choose these sets to contain more elements, allowing μ_n^* and μ_{n+1}^* to

range over a larger portion of the space, and allowing the KG factor calculation to more accurately approximate the value that would result if we implemented the best option. However, the trade-off is that as we increase the size of these sets, computing the KG factor is slower, making implementation of the KG method more computationally intensive.

1.3 Bayesian Optimization for Cheminformatics

Cheminformatics is an interdisciplinary field that combines chemistry, computer science and information science, and has received much attention in the past decade. The primary application of Cheminformatics includes virtual screening of libraries of chemical compounds for identifying functional molecules [49, 80]. For example, pharmaceutical companies use the virtual screening techniques to preselect and narrow down candidate compounds in the process of drug discovery.

Machine learning plays a substantial role in Cheminformatics, and in particular, the researchers can use machine learning prediction of the molecular activities, to efficiently guide their experimental evaluation in the search for the functional molecules of interest. In this context, machine learning researchers have compared two approaches: (1) evaluate the molecules with the best predicted performance according to the trained machine learning model (the so-called "pure exploitation" approach); (2) evaluate the molecules having the most potential to improve over the best previously evaluated molecule (the Bayesian optimization approach). Although the Bayesian optimization approach has shown its superior performance over the "pure exploitation" approach in a variety of experimental design applications [26, 9, 86], Bayesian optimization has been pursued only in a limited way

in Cheminformatics [66, 56], the majority of work leveraging machine learning for searching functional molecules has used pure exploitation [2, 81, 6].

To apply Bayesian optimization to a broader range of Cheminformatics applications, we need additional work: first, the Gaussian process regression method we have discussed in Section 1.2.1 assumes the objective function is continuous, while in Cheminformatics applications, the search space is usually discrete. This problem has been addressed by [66], where they proposed a linear additive model to predict the measure of molecular activity. However, the sampling policy in this work is only computationally feasible for a limited number of candidates (typically less than 10^7). Second, the response value is not always a scalar. For instance, high throughput screening, a popular technique in drug screening, reports whether the candidate molecule is "active hit" or not, which is a binary response. Third, it is incredibly common to examine hundreds to thousands of molecules in parallel in Cheminformatics applications. Although there are a few works in Bayesian optimization with applications to other fields [86, 26, 94], and work in active learning [9], that proposed parallel sampling strategies, none of the methods handles discrete function domain or binary response. To this end, we propose a novel discrete Bayesian optimization method, which we will discuss in Chapter 3.

1.4 Thesis Organization

This thesis is built upon the author's previously published works: the review of Bayesian optimization in Section 1.2 can be found in [21]; Chapter 2 is based on [94]; Chapter 3 is based on a working paper by the author, which is being prepared for journal submission at the time of completing this thesis; Chapter 4 is

based on [69] and [70]. We now give an overview of each chapter as follows:

Chapter 2

Chapter 2 considers parallel Bayesian optimization, and proposes an efficient method based on stochastic approximation for implementing a conceptual Bayesian optimization algorithm proposed by [26]. To accomplish this, Chapter 2 uses infinitesimal perturbation analysis (IPA) to construct a stochastic gradient estimator and shows that this estimator is unbiased. Chapter 2 also shows that the stochastic gradient ascent algorithm using the constructed gradient estimator converges to a stationary point of the *q-EI* surface, and therefore, as the number of multiple starts of the gradient ascent algorithm and the number of steps for each start grow large, the one-step Bayes optimal set of points is recovered. Chapter 2 shows in numerical experiments that the method for maximizing the *q-EI* is faster than methods based on closed-form evaluation using high-dimensional integration, when considering many parallel function evaluations, and is comparable in speed when considering few. Chapter 2 also shows that the resulting one-step Bayes optimal algorithm for parallel global optimization finds high quality solutions with fewer evaluations than a heuristic based on approximately maximizing the *q-EI*. A high quality open source implementation of this algorithm is available in the open source Metrics Optimization Engine (MOE).

Chapter 3

Chapter 3 consider a discrete Bayesian optimization problem arising in biochemistry, in which we wish to find a peptide that (1) has a certain expensive-to-ascertain biochemical property (it is a substrate for two protein-modifying en-

zymes, phosphopantetheinyltransferase and ACP hydrolase); and (2) is as short as possible. Finding such a peptide would allow tracking protein interactions with great ease, and would support a number of innovations in medicine, biochemistry, and materials science. However, such peptides are difficult to find, because the set of peptides is large, only a small fraction have the desired property, and ascertaining whether or not a peptide has the desired property requires performing a time-consuming laboratory experiment. Chapter 3 presents a novel Bayesian optimization method for choosing which peptides to test to find such a peptide as quickly as possible. Chapter 3 proves theoretical bounds on its solution quality, demonstrates in simulation that it outperforms two natural benchmark methods, and then describes how it was used in practice to find a peptide with the desired property that is shorter than the shortest previously known. While the method was developed for this specific application in biochemistry, it can also be used in other discrete Bayesian optimization problems in which we wish to find an exemplar whose expensive-to-obtain binary label is positive, and for which a secondary easy-to-evaluate cost objective is as small as possible.

Chapter 4

Chapter 4 considers two closely related problems in Bayesian optimization: the first is called *multi-information source optimization problem*, or MISO, where the goal is to optimize an expensive-to-evaluate black-box function, and in addition to evaluating the objective function directly, there are multiple cheap approximate estimates of the objective that we could use, the so-called "information sources". This scenario typically arises in engineering sciences. For example, in aerospace engineering, when designing an airfoil, there are various computer simulators, based on different physical models and with different fidelity, to evaluate the perfor-

mance of a design. The computer simulators in this context, are information sources, where some of them are more computationally expensive and may provide more accurate estimate, while the cheaper information sources may only offer a crude estimate with much less computational cost. The goal is to efficiently utilize the information sources and reduce the overall cost during the optimization task. The second problem is called warm starting Bayesian optimization, which aims to speedup the optimization routine using additional information from previously solved related problems. This scenario typically occurs when one solves a series of optimization problems, where the objective functions are from the same family but with variations. For example, decision making problems that use one optimization model and input data collected over different time periods, are interrelated.

Both problems are similar in that there are multiple "sources" that are closely related to the optimization objective, and are cheaper to query than the original objective. Chapter 4 proposes a novel Bayesian model that specifically analyzes the correlation structure among the "sources", and then presents a cost-sensitive Knowledge Gradient algorithm that makes optimal decision in choosing both which point to sample and using which "source".

Chapter 5

Chapter 5 summarizes the contributions of this thesis and describes ongoing work in Bayesian optimization known to this author. In particular, it describes prospective problems in biochemistry and engineering, where the Bayesian optimization techniques developed in this thesis are being used or have potential usage, and new methodological work in parallel Bayesian optimization and multi-information source Bayesian optimization.

CHAPTER 2

PARALLEL BAYESIAN OPTIMIZATION OF EXPENSIVE FUNCTIONS

2.1 Introduction

The traditional Bayesian optimization procedures, including the methods we have discussed in Chapter 1, are all sequential, which means we iteratively sample one point at a time. In the recent years, with the advance of technology in many areas, parallelization is becoming increasingly popular, and sometimes crucial to the feasibility of a problem. For example, in computational science and engineering, the advent of multi-core computing speeds up the computation in wall clock time, making it essential in time sensitive tasks. In drug discovery, the research lab needs to make a lot of candidate molecules and test individually, which is extremely time consuming; with the introduction of robotics and automation, one can test hundreds of candidates at the same time, and speed up the process significantly. Then it is natural to ask how we can parallelize our Bayesian optimization procedures. This chapter proposes a novel parallel Bayesian optimization method, which is a generalization of expected improvement to its parallel setting.

The expected improvement algorithm, known as `EGO` [42], chooses each point at which to evaluate the expensive objective function in the "outer" expensive global optimization problem, i.e., the problem in (1.1), by solving an "inner" optimization problem: maximize expected improvement. If this is the final point that will be evaluated in the outer optimization problem, and if additional conditions are satisfied (the evaluations are free from noise, and the implementation decision, i.e., the solution that will be implemented in practice after the optimization is

complete, is restricted to be a previously evaluated point), then the point with largest expected improvement is the Bayes-optimal point to evaluate, in the sense of providing the best possible average-case performance in the outer expensive global optimization problem [21].

The notion of expected improvement was generalized by [26] to the parallel setting, in which the expensive objective can be evaluated at several points simultaneously. This generalization, called the "multi-points expected improvement" or the *q-EI*, is consistent with the decision-theoretic derivation of expected improvement, and quantifies the expected utility that will result from the evaluation of a *set* of points. [26] also provided an analytical formula for the case when $q = 2$, or 2-*EI*.

If the generalized inner optimization problem proposed by [26], which is to find the set of points to evaluate next that jointly maximize the *q-EI*, could be solved efficiently, then this would provide the one-step Bayes-optimal set of points to evaluate in the outer problem, and would create a one-step Bayes-optimal algorithm for global optimization of expensive functions able to fully utilize parallelism.

This generalized inner optimization problem is challenging, however, because unlike the (scalar) expected improvement used by `EGO`, which has an easy-to-compute and easy-to-differentiate expression as shown in (1.11), the *q-EI* lacks an easy-to-compute expression, and is calculable only through Monte Carlo simulation, high-dimensional numerical integration, or expressions involving high-dimensional multivariate normal cumulative distribution functions. This significantly restricts the set of applications in which a naive implementation can solve the inner problem faster than a single evaluation of the outer optimization problem. Stymied by this difficulty, [26], as well as the later work in [10], propose heuristic

methods that are *motivated* by the one-step optimal algorithm of evaluating the set of points that jointly maximize the *q-EI*, but that do not actually achieve this gold standard.

Contributions. The main contribution of the research described in this chapter is to provide a method that solves the inner optimization problem of maximizing the *q-EI* efficiently, creating a practical and broadly applicable one-step Bayes-optimal algorithm for parallel global optimization of expensive functions. To accomplish this we use infinitesimal perturbation analysis (IPA) (see [33]) to construct a stochastic gradient estimator of the gradient of the *q-EI* surface, and show that this estimator is unbiased, with a bounded second moment. Our method uses this estimator within a stochastic gradient ascent algorithm, and we show that it converges to a stationary point of the *q-EI* surface. We use multiple restarts to identify multiple stationary points, and then use ranking and selection to identify the best stationary point found. As the number of restarts and the number of iterations of stochastic gradient ascent within each start both grow large, the one-step optimal set of points to evaluate is recovered.

Our method can be implemented in both synchronous environments, in which function evaluations are performed in batches and finish at the same time, and asynchronous ones, in which a function evaluation may finish before others are done.

In addition to our methodological contribution, we have developed a high-quality open source software package, the "Metrics Optimization Engine (MOE)" [12], implementing our method for solving the inner optimization problem, and the resulting algorithm for parallel global optimization of expensive functions. To fur-

ther enhance computational speed, the implementation takes advantage of parallel computing, and achieves 100X speedup over single-threaded computation when deployed on a graphical processing unit (GPU). This software package has been used by Yelp and Netflix to solve global optimization problems arising in their businesses [11, 4]. For the rest of this chapter, we refer to our method as "MOE-qEI" because it is implemented in MOE.

We compare MOE-qEI against several benchmark methods. We show that MOE-qEI provides high-quality solutions to the outer optimization problem more quickly than the heuristic CL-mix policy proposed by [10], which is motivated by the inner optimization problem. We also show that MOE-qEI provides a substantial parallel speedup over the single-threaded `EGO` algorithm, which is one-step optimal when parallel resources are unavailable. We also compare our simulation-based method for solving the inner optimization problem against methods based on exact evaluation of the *q-EI* from [10] and [61] (discussed in more detail below) and show that our simulation-based approach to solving the inner optimization problem provides solutions to both the inner and outer optimization problem that are comparable in quality and speed when q is small, and superior when q is large.

Related Work. Developed independently and in parallel with this work is [10], which provides a closed-form formula for computing *q-EI*, and the book chapter [61], which provides a closed-form expression for its gradient. Both require multiple calls to high-dimensional multivariate normal cumulative distribution functions (cdfs). These expressions can be used within an existing continuous optimization algorithm to solve the inner optimization problem that we consider.

While attractive in that they provide closed-form expressions, calculating these

expressions when q is even moderately large is slow and numerically challenging. This is because calculating the multivariate normal cdf in moderately large dimension is itself challenging, with state of the art methods relying on numerical integration or Monte Carlo sampling as described in [23]. Indeed, the method for evaluating the *q-EI* from [10] requires q^2 evaluations of the $q-1$ dimensional multivariate normal cdf, and the method for evaluating its gradient requires $O(q^4)$ calls to multivariate normal cdfs with dimension ranging from $q-3$ to q. In our numerical experiments, we demonstrate that our method for solving the inner optimization problem requires less computation time and parallelizes more easily than do these competing methods, for $q > 4$, and performs comparably when q is smaller. We also demonstrate that MOE-qEI's improved performance in the inner optimization problem for $q > 4$ translates to improved performance in the outer optimization problem.

Other related work includes the previously proposed heuristic CL-mix from [10], which does not solve the inner maximization of *q-EI*, instead using an approximation. While solving the inner maximization of *q-EI* as we do makes it more expensive to compute the set of points to evaluate next, we show in our numerical experiments that it results in a substantial savings in the number of evaluations required to find a point with a desired quality. When function evaluations are expensive, this results in a substantial reduction in overall time to reach an approximately optimal solution.

In other related work on parallel Bayesian optimization, [22] and [99] proposed a Bayesian optimization algorithm that evaluates pairs of points in parallel, and is one-step Bayes-optimal in the noisy setting under the assumption that one can only observe noisy function values for single points, or noisy function value differences

between pairs of points. This algorithm, however, is limited to evaluating pairs of points, and does not extend to a higher level of parallelism.

There are also other non-Bayesian algorithms for derivative-free global optimization of expensive functions with parallel function evaluations from [13, 46] and [36]. These are quite different in spirit from the algorithm we develop, not being derived from a decision-theoretic foundation.

Outline of this Chapter. With Gaussian Process regression and expected improvement already been discussed in Chapter 1, we begin in Section 2.2 by defining the *q-EI* and the one-step optimal algorithm. We construct our stochastic gradient in Section 2.3.2, and combine this estimator together with stochastic gradient ascent to define a one-step optimal method for parallel Bayesian global optimization in Section 2.3.3. Then in Section 2.4.1 we show that the constructed gradient estimator of the *q-EI* surface is unbiased under certain mild regularity conditions. Moreover, in Section 2.4.2 we provide convergence analysis of the stochastic gradient ascent algorithm. Finally, in Section 2.5 we present numerical experiments: we compare MOE-qEI against previously proposed heuristics from the literature; we demonstrate that MOE-qEI provides a speedup over single-threaded `EGO`; we show that MOE-qEI is more efficient than optimizing evaluations of the *q-EI* using closed-form formula provided in [10] when q is large; and we show that MOE-qEI computes the gradient of *q-EI* faster than evaluating the closed-form expression proposed in [61].

2.2 Multi-points Expected Improvement (*q-EI*)

In a parallel computing environment, we wish to use the posterior distribution given by (1.5) to choose the set of points to evaluate next. [26] proposed making this choice using a decision-theoretic approach, in which we consider the utility that evaluating a particular candidate set of points would provide, in terms of their ability to reveal points with objective function values better than previously known. We review this decision-theoretic approach here, and then present a new algorithm for implementing this choice in the next section.

Let q be the number of function evaluations that we may perform in parallel, and let $\boldsymbol{X}$ be a candidate set of points that we are considering evaluating next. Let $f_n^\star = \max_{m \leq n} f(\boldsymbol{x}^{(m)})$ indicate the value of the best point evaluated, before beginning these q new function evaluations. The value of the best point evaluated after all q function evaluations are complete will be $\max\left(f_n^\star, \max_{i=1,\ldots,q} f(\boldsymbol{x}_i)\right)$. The difference between these two values (the values of the best point evaluated, before and after these q new function evaluations) is called the *improvement*, and is equal to $\left(\max_{i=1,\ldots,q} f(\boldsymbol{x}_i) - f_n^\star\right)^+$, where $a^+ = \max(a, 0)$ for $a \in \mathbb{R}$.

We then value a joint set of evaluations at these candidate points $\boldsymbol{X}$ as the expected value of this improvement, and we refer to this quantity as the *multi-points expected improvement* or *q-EI* from [26]. This multi-points expected improvement can be written as,

$$q\text{-}EI(\boldsymbol{X}) = \mathbb{E}_n\left[\left(\max_{i=1,\ldots,q} f(\boldsymbol{x}_i) - f_n^\star\right)^+\right], \tag{2.1}$$

where $\mathbb{E}_n[\cdot] := \mathbb{E}\left[\cdot | \boldsymbol{x}^{(1:n)}, y^{(1:n)}\right]$ is the expectation taken with respect to the posterior distribution.

[26] then proposes that we should choose to next evaluate the set of points that maximizes the multi-points expected improvement,

$$\underset{\boldsymbol{X} \in \mathbb{A}^q}{\operatorname{argmax}}\, q\text{-}EI(\boldsymbol{X}). \tag{2.2}$$

In the special case $q = 1$, which occurs when we are operating without parallelism, the multi-points expected improvement reduces to the expected improvement, as considered by [63] and [42], and can be evaluated in closed-form, in terms of the normal pdf and cdf. The algorithm that chooses the next point to evaluate according to (2.2) is the `EGO` algorithm of [42].

[26] provided an analytical calculation of `EI` when $q = 2$, but in the same paper Ginsbourger commented that the general case of *q-EI* has complex expressions depending on q-dimensional Gaussian cumulative distribution functions, and computation of *q-EI* when q is large would have to rely on numerical multivariate integral approximation techniques, which is computationally expensive and makes solving (2.2) difficult. [25] writes "directly optimizing the *q-EI* becomes extremely expensive as q and d (the dimension of inputs) grow".

2.3 Algorithm

In this section we present a new algorithm for solving the inner optimization problem (2.2) of maximizing *q-EI*. This algorithm uses a novel estimator of the gradient of the *q-EI* presented in Section 2.3.2, used within a multistart stochastic gradient ascent framework as described in Section 2.3.3. We additionally generalize this technique from synchronous to asynchronous parallel optimization is Section 2.3.4. We begin by introducing some additional notation used to describe our algorithm.

2.3.1 Notation

In this section we reformulate previously defined equations and define additional notation to better support construction of the gradient estimator.

We first write the posterior distribution on $f(\boldsymbol{X})$ given by (1.5) in an alternative expression

$$f(\boldsymbol{X}) \overset{d}{=} \boldsymbol{\mu}(\boldsymbol{X}) + \boldsymbol{L}(\boldsymbol{X})\boldsymbol{Z}, \tag{2.3}$$

where $\boldsymbol{L}(\boldsymbol{X})$ is the lower triangular matrix obtained from the Cholesky decomposition of $\boldsymbol{\Sigma}^{(n)}$ in (1.5), $\boldsymbol{\mu}(\boldsymbol{X})$ is the posterior mean (identical in meaning to $\boldsymbol{\mu}^{(n)}$ in (1.5), but rewritten here to emphasize the dependence on X and deemphasize the dependence on n), and $\boldsymbol{Z}$ is a q-dimensional standard normal random vector.

By substituting (2.3) into (2.1), we have

$$q\text{-}EI(\boldsymbol{X}) = \mathbb{E}\left[\left(\max_{i=1,\ldots,q} \boldsymbol{e}_i \left[\boldsymbol{\mu}(\boldsymbol{X}) + \boldsymbol{L}(\boldsymbol{X})\boldsymbol{Z}\right] - f_n^*\right)^+\right], \tag{2.4}$$

where $\boldsymbol{e}_i$ is a unit vector in direction i and the expectation is over $\boldsymbol{Z}$. To make (2.4) even more compact, define a new vector $\boldsymbol{m}(\boldsymbol{X})$ and new matrix $\boldsymbol{C}(\boldsymbol{X})$,

$$\begin{aligned} \boldsymbol{m}(\boldsymbol{X})_i &= \begin{cases} \boldsymbol{\mu}(\boldsymbol{X})_i - f_n^* & \text{if } i > 0\ , \\ 0 & \text{if } i = 0\ , \end{cases} \\ \boldsymbol{C}(\boldsymbol{X})_{ij} &= \begin{cases} \boldsymbol{L}(\boldsymbol{X})_{ij} & \text{if } i > 0\ , \\ 0 & \text{if } i = 0\ , \end{cases} \end{aligned} \tag{2.5}$$

and (2.4) becomes

$$q\text{-}EI(\boldsymbol{X}) = \mathbb{E}\left[\max_{i=0,\ldots,q} \boldsymbol{e}_i \left[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}\right]\right]. \tag{2.6}$$

2.3.2 Constructing the Gradient Estimator

We now construct our estimator of the gradient $\nabla q\text{-}EI(\boldsymbol{X})$. Let

$$h(\boldsymbol{X}, \boldsymbol{Z}) = \max_{i=0,\ldots,q} \boldsymbol{e}_i \left[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}\right]. \tag{2.7}$$

Then

$$\nabla q\text{-}EI(\boldsymbol{X}) = \nabla \mathbb{E} h(\boldsymbol{X}, \boldsymbol{Z}). \tag{2.8}$$

If gradient and expectation in (2.8) is interchangeable, the gradient would be

$$\nabla q\text{-}EI(\boldsymbol{X}) = \mathbb{E}\boldsymbol{g}(\boldsymbol{X}, \boldsymbol{Z}), \tag{2.9}$$

where

$$\boldsymbol{g}(\boldsymbol{X}, \boldsymbol{Z}) = \begin{cases} \nabla h(\boldsymbol{X}, \boldsymbol{Z}) & \text{if } \nabla h(\boldsymbol{X}, \boldsymbol{Z}) \text{ exists,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

$\boldsymbol{g}(\boldsymbol{X}, \boldsymbol{Z})$ can be computed using results on differentiation of the Cholesky decomposition from [83]. We then propose to use $\boldsymbol{g}(\boldsymbol{X}, \boldsymbol{Z})$ as our estimator of the gradient $\nabla q\text{-}EI$, and will discuss interchangeability of gradient and expectation, which implies unbiasedness of our gradient estimator, in Section 2.4.1. As will be discussed in Section 2.4.2, unbiasedness of the gradient estimator is one of the sufficient conditions for convergence of the stochastic gradient ascent algorithm proposed in Section 2.3.3.

2.3.3 Optimization of q-EI

Our stochastic gradient ascent algorithm begins with some initial point $\boldsymbol{X}_0 \in \mathbb{A}^q$, and generates a sequence $\{\boldsymbol{X}_t : t = 1, 2, \ldots\}$ using

$$\boldsymbol{X}_{t+1} = \prod_{\mathbb{A}^q} \left[\boldsymbol{X}_t + \epsilon_t \boldsymbol{G}(\boldsymbol{X}_t)\right], \tag{2.11}$$

where $\prod_{\mathbb{A}^q}(\boldsymbol{X})$ denotes the closest point in $\mathbb{A}^q$ to $\boldsymbol{X}$, and if the closest point is not unique, we use a closest point such that the function $\prod_{\mathbb{A}^q}(\cdot)$ is measurable. Here, we restrict our search for the maximum of the *q-EI* to $\mathbb{A}^q$ since the feasible set of the outer optimization problem is $\mathbb{A}$. One could also define a more restricted search region than $\mathbb{A}^q$, and revise (2.11) accordingly. $\boldsymbol{G}(\boldsymbol{X}_t)$ is an estimate of the gradient of *q-EI*$(\cdot)$ at $\boldsymbol{X}_t$. In our implementation of the algorithm, we use Monte Carlo simulation to estimate the gradient, averaging together M replicates of our stochastic gradient estimator,

$$\boldsymbol{G}(\boldsymbol{X}_t) = \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{g}(\boldsymbol{X}_t, \boldsymbol{Z}_{t,m}), \tag{2.12}$$

where $\{\boldsymbol{Z}_{t,m}$: m=1, ..., M$\}$ are i.i.d. samples generated from the q-dimensional standard normal distribution, and $\boldsymbol{g}(\boldsymbol{X}_t, \boldsymbol{Z}_{t,m})$ is defined in (2.10). $\{\epsilon_t : t = 0, 1, \ldots\}$ is a stochastic gradient stepsize sequence [52], typically chosen to be equal to $\epsilon_t = \frac{a}{t^\gamma}$ for some scalar a and γ.

To find the global maximum of the *q-EI*, we use multiple restarts of the algorithm from a set of starting points, drawn from a Latin hypercube design [62], to find multiple stationary points, and then use simulation, written here with a fixed sample size N for simplicity, to identify the best stationary point found. We summarize the procedures in Algorithm 2.1.

Algorithm 2.1. *Optimization of q-EI*

Require: *number of starting points R; stepsize constants a and γ; number of steps for one run of gradient ascent T; number of Monte Carlo samples for estimating the gradient M; number of Monte Carlo samples for estimating q-EI N.*

1: *Draw R starting points from a Latin hypercube design in $\mathbb{A}^q$, $\boldsymbol{X}_{r,0}$ for $r = 1, \ldots, R$.*

2: ***for*** $r = 1$ *to* R ***do***

3: ***for*** $t = 0$ *to* $T - 1$ ***do***

4: *Compute* $\boldsymbol{G}_t = \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{g}(\boldsymbol{X}_{r,t}, \boldsymbol{Z}_{r,t,m})$ *where* $\boldsymbol{Z}_{r,t,m}$ *is a vector of* q *i.i.d. samples drawn from the standard normal distribution.*

5: *Update solution using stochastic gradient ascent* $\boldsymbol{X}_{r,t+1} = \boldsymbol{X}_{r,t} + \frac{a}{t^\gamma}\boldsymbol{G}_t$.

6: ***end for***

7: *Estimate* q*-EI*$(\boldsymbol{X}_{r,T})$ *using Monte Carlo simulation with* N *i.i.d. samples, and store the estimate as* $\widehat{q\text{-}EI}_r$.

8: ***end for***

9: ***return*** $\boldsymbol{X}_{r',T}$ *where* $r' = \text{argmax}_{r=1,\ldots,R}\, \widehat{q\text{-}EI}_r$.

One could replace this simple comparison based on N Monte Carlo samples with a more sophisticated ranking and selection algorithm discussed in [48]. We leave this out of Algorithm 2.1 for simplicity of description. One can also replace estimation of *q-EI* in Step 7 by a method using closed-form formula in [10].

In our implementation of this algorithm, we supply optional fallback logic. This fallback logic takes two additional parameters: a strictly positive real number ϵ', and an integer L. If $\max_{i=1,\ldots,R} \widehat{q\text{-}EI}_r \leq \epsilon'$, so that multistart stochastic gradient ascent failed to find a point with estimated expected improvement better than ϵ', then we generate L additional solutions from a Latin Hypercube on $\mathbb{A}^q$, estimate the expected improvement at each of these using the same Monte Carlo approach as in Step 7, and select the one with the largest estimated expected improvement.

Analysis in Section 2.4.2 shows that under certain conditions, including unbiasedness of the gradient estimator, the stochastic gradient algorithm converges to a stationary point almost surely.

2.3.4 Asynchronous Parallel Optimization

(2.2) corresponds to synchronous parallel optimization, in which we wait for all q points from our previous batch to finish before searching for a new set of points. However, in some applications, we may wish to generate a new partial batch of points to evaluate next while p points are still being evaluated, before we have their values. This is common in expensive computer simulations, where simulations do not necessarily finish at the same time.

We can extend (2.2) to asynchronous parallel optimization: suppose parallelization allows a batch of q points to evaluate simultaneously; the first p points are still under evaluation, while the remaining $q - p$ points have finished evaluation and the resources used to evaluate them are free to evaluate new points. We let $\boldsymbol{X'} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$ be the first p points still under evaluation, and let $\boldsymbol{X} := (\boldsymbol{x}_{p+1}, \ldots, \boldsymbol{x}_q)$ be the $(q-p)$ points ready for new evaluations. Computation of *q-EI* for these q points remains the same as in (2.1), but we use an alternative notation, *q-EI*$(\boldsymbol{X'}, \boldsymbol{X})$, to explicitly indicate that $\boldsymbol{X'}$ are the points still being evaluated and $\boldsymbol{X}$ are the new points to evaluate. Keeping $\boldsymbol{X'}$ fixed, we optimize *q-EI* over $\boldsymbol{X}$ by solving this alternative problem

$$\underset{\boldsymbol{X} \subset \mathbb{A}}{\operatorname{argmax}}\ q\text{-}EI(\boldsymbol{X'}, \boldsymbol{X}). \tag{2.13}$$

As we did in the algorithm for synchronous parallel optimization in Section 2.3.3, we estimate the gradient of the objective function with respect to $\boldsymbol{X}$, i.e., $\nabla_{\boldsymbol{X}}\, q\text{-}EI(\boldsymbol{X'}, \boldsymbol{X})$. The gradient estimator is essentially the same as in Section 2.3.2, except that we only take derivatives of $h(\cdot, \cdot)$ with respect to $\boldsymbol{X}$. Then we proceed the same way as in Algorithm 2.1.

2.4 Theoretical Analysis

In Section 2.3, when we constructed our gradient estimator and described the use of stochastic gradient ascent to optimize *q-EI*, we alluded to conditions under which this gradient estimator is unbiased and this algorithm converges to a stationary point of the *q-EI* surface. In this section, we describe these conditions and show unbiasedness and almost sure convergence to a stationary point of the *q-EI*.

2.4.1 Unbiasedness of the Gradient Estimator

We present a pair of lemmas and then our main theorem for unbiasedness of the gradient estimator. The lemmas will be used to support the proof of the theorem.

Lemma 2.1. *If a function* $\Psi : \mathbb{R} \mapsto \mathbb{R}$ *is twice continuously differentiable over* $[-\epsilon, \epsilon]$*, where we define* $0/0 = 0$*, then for a sequence* $(\delta_\ell) \subseteq [-\epsilon, \epsilon]$*,*

$$\sup_\ell \left| \frac{\Psi(\delta_\ell) - \Psi(0)}{\delta_\ell} \right| < \infty.$$

Proof. By Taylor's theorem,

$$\Psi(\delta_\ell) = \Psi(0) + \Psi'(0)\delta_\ell + \frac{\Psi''(r_\ell)}{2}\delta_\ell^2,$$

where r_ℓ is between 0 and δ_ℓ. Then

$$\begin{aligned} \sup_\ell \left| \frac{\Psi(\delta_\ell) - \Psi(0)}{\delta_\ell} \right| &= \sup_\ell \left| \Psi'(0) + \frac{\Psi''(r_\ell)}{2}\delta_\ell \right|, \\ &\leq |\Psi'(0)| + \sup_\ell \left| \frac{\Psi''(r_\ell)}{2}\delta_\ell \right| \quad \text{(by the triangle inequality).} \end{aligned}$$

Since $\delta_\ell \in [-\epsilon, \epsilon]$, then $r_\ell \in [-\epsilon, \epsilon]$. We also have $\Psi''(\cdot)$ continuous over $[-\epsilon, \epsilon]$, thus

$$\sup_\ell \left| \frac{\Psi''(r_\ell)}{2}\delta_\ell \right| \leq \epsilon \sup_\ell \left| \frac{\Psi''(r_\ell)}{2} \right| < \infty.$$

□

Lemma 2.2. *If $\boldsymbol{m}(\boldsymbol{X})$ and $\boldsymbol{C}(\boldsymbol{X})$ are differentiable in a neighborhood of $\boldsymbol{X}$, and there are no duplicated rows in $\boldsymbol{C}(\boldsymbol{X})$, then $\nabla h(\boldsymbol{X}, \boldsymbol{Z})$ exists almost surely for any $\boldsymbol{X}$.*

Proof. Observe that

$$h(\boldsymbol{X}, \boldsymbol{Z}) = \boldsymbol{e}_{I^*} \left[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}\right],$$

where $I^* \in \text{argmax}_{i=0,\ldots,q} \boldsymbol{e}_i \left[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}\right] := \mathcal{S}$.

We claim that $\nabla h(\boldsymbol{X}, \boldsymbol{Z})$ exists only if $\boldsymbol{e}_I(\frac{\partial \boldsymbol{m}(\boldsymbol{X})}{\partial x_{ik}} + \frac{\partial \boldsymbol{C}(\boldsymbol{X})}{\partial x_{ik}}\boldsymbol{Z})$ are equal $\forall I \in \mathcal{S}$, and $\forall i, k$ (k iterates over dimension). Therefore,

$$\begin{aligned}
\mathbb{P}(\nabla h(\boldsymbol{X}, \boldsymbol{Z}) \text{ does not exist}) &\leq \mathbb{P}(|\mathcal{S}| \geq 2), \\
&\leq \frac{1}{2} \sum_{i \neq j} \mathbb{P}\left(\boldsymbol{e}_i[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}] = \boldsymbol{e}_j[\boldsymbol{m}(\boldsymbol{X}) + \boldsymbol{C}(\boldsymbol{X})\boldsymbol{Z}]\right), \\
&= \frac{1}{2} \sum_{i \neq j} \mathbb{P}\left((\boldsymbol{C}(\boldsymbol{X})_{i\cdot} - \boldsymbol{C}(\boldsymbol{X})_{j\cdot})\,\boldsymbol{Z} = m(\boldsymbol{X})_j - m(\boldsymbol{X})_i\right),
\end{aligned}$$

where $\boldsymbol{C}(\boldsymbol{X})_{i\cdot}$ is ith row of $\boldsymbol{C}(\boldsymbol{X})$.

Since $\boldsymbol{C}(\boldsymbol{X})_{i\cdot} \neq \boldsymbol{C}(\boldsymbol{X})_{j\cdot}$, $\{\boldsymbol{Z} : (\boldsymbol{C}(\boldsymbol{X})_{i\cdot} - \boldsymbol{C}(\boldsymbol{X})_{j\cdot})\,\boldsymbol{Z} = m(\boldsymbol{X})_j - m(\boldsymbol{X})_i\}$ is subspace of $\mathbb{R}^q$ with dimension smaller than q, thus

$$\mathbb{P}\left((\boldsymbol{C}(\boldsymbol{X})_{i\cdot} - \boldsymbol{C}(\boldsymbol{X})_{j\cdot})\,\boldsymbol{Z} = m(\boldsymbol{X})_j - m(\boldsymbol{X})_i\right) = 0 \quad \forall i \neq j.$$

Hence

$$\mathbb{P}(\nabla h(\boldsymbol{X}, \boldsymbol{Z}) \text{ does not exist}) \leq 0 = 0.$$

□

Theorem 2.1. *If $\boldsymbol{m}(\boldsymbol{X})$ and $\boldsymbol{C}(\boldsymbol{X})$ are twice continuously differentiable in a neighborhood of $\boldsymbol{X}$, and $\boldsymbol{C}(\boldsymbol{X})$ has no duplicate rows, then $\nabla h(\boldsymbol{X}, \boldsymbol{Z})$ exists almost surely and*

$$\nabla \mathbb{E} h(\boldsymbol{X}, \boldsymbol{Z}) = \mathbb{E} \nabla h(\boldsymbol{X}, \boldsymbol{Z}).$$

Proof. Without loss of generality, we only look at perturbation on kth dimension of mth point $\boldsymbol{x}_m$ within $\boldsymbol{X}$, where $m = 1, \ldots, q$. We use $\boldsymbol{e}_{m,k}$ to denote direction of perturbation, and let δ be the magnitude of perturbation, then the new set of points after perturbation is $\boldsymbol{X} + \delta \boldsymbol{e}_{m,k}$. In a slight abuse of notation, we define

$$\begin{aligned}
h(\delta, \boldsymbol{Z}) &:= h(\boldsymbol{X} + \delta \boldsymbol{e}_{m,k}, \boldsymbol{Z}), \\
\boldsymbol{m}(\delta) &:= \boldsymbol{m}(\boldsymbol{X} + \delta \boldsymbol{e}_{m,k}), \\
\boldsymbol{C}(\delta) &:= \boldsymbol{C}(\boldsymbol{X} + \delta \boldsymbol{e}_{m,k}), \\
I^*_{(\delta, \boldsymbol{Z})} &:= \min \left(\underset{i=0,\ldots,q}{\operatorname{argmax}} \boldsymbol{e}_i \left[\boldsymbol{m}(\delta) + \boldsymbol{C}(\delta) \boldsymbol{Z} \right] \right).
\end{aligned}$$

Then

$$h(\delta, \boldsymbol{Z}) = \max_{i=0,\ldots,q} \boldsymbol{e}_i \left[\boldsymbol{m}(\delta) + \boldsymbol{C}(\delta) \boldsymbol{Z} \right] = \boldsymbol{e}_{I^*_{(\delta, \boldsymbol{Z})}} \left[\boldsymbol{m}(\delta) + \boldsymbol{C}(\delta) \boldsymbol{Z} \right].$$

We also define

$$\begin{aligned}
\triangle(\delta, \boldsymbol{Z}) :&= h(\delta, \boldsymbol{Z}) - h(0, \boldsymbol{Z}), \\
&= \boldsymbol{e}_{I^*_{(\delta, \boldsymbol{Z})}} \left[\boldsymbol{m}(\delta) + \boldsymbol{C}(\delta) \boldsymbol{Z} \right] - \boldsymbol{e}_{I^*_{(0, \boldsymbol{Z})}} \left[\boldsymbol{m}(0) + \boldsymbol{C}(0) \boldsymbol{Z} \right].
\end{aligned}$$

Let $\epsilon > 0$. Consider a sequence $(\delta_\ell) \subseteq [-\epsilon, \epsilon]$ and $\delta_\ell \searrow 0$ as $\ell \to \infty$. We want to show $\lim_{\ell \to \infty} \frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell}$ exists almost surely, and

$$\lim_{\ell \to \infty} \mathbb{E} \left[\frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell} \right] = \mathbb{E} \left[\lim_{\ell \to \infty} \frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell} \right].$$

As a first step we show that $\sup_\ell \left| \frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell} \right|$ is bounded. For any δ in the sequence (δ_ℓ), we consider 2 cases,

- Case 1: If $\boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)+\boldsymbol{C}(\delta)\boldsymbol{Z}\right] \geq \boldsymbol{e}_{I^*_{(0,\boldsymbol{Z})}}\left[\boldsymbol{m}(0)+\boldsymbol{C}(0)\boldsymbol{Z}\right]$, then

$$\begin{aligned}
|\triangle(\delta,\boldsymbol{Z})| &= \boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)+\boldsymbol{C}(\delta)\boldsymbol{Z}\right] - \boldsymbol{e}_{I^*_{(0,\boldsymbol{Z})}}\left[\boldsymbol{m}(0)+\boldsymbol{C}(0)\boldsymbol{Z}\right], \\
&\leq \boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)+\boldsymbol{C}(\delta)\boldsymbol{Z}\right] - \boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(0)+\boldsymbol{C}(0)\boldsymbol{Z}\right], \\
&= \boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)-\boldsymbol{m}(0)+\boldsymbol{C}(\delta)\boldsymbol{Z}-C(0)\boldsymbol{Z}\right], \\
&\leq \left|\boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)-\boldsymbol{m}(0)+\boldsymbol{C}(\delta)\boldsymbol{Z}-C(0)\boldsymbol{Z}\right]\right|.
\end{aligned}$$

- Case 2: If $\boldsymbol{e}_{I^*_{(\delta,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)+\boldsymbol{C}(\delta)\boldsymbol{Z}\right] \leq \boldsymbol{e}_{I^*_{(0,\boldsymbol{Z})}}\left[\boldsymbol{m}(0)+\boldsymbol{C}(0)\boldsymbol{Z}\right]$, then

$$|\triangle(\delta,\boldsymbol{Z})| \leq \left|\boldsymbol{e}_{I^*_{(0,\boldsymbol{Z})}}\left[\boldsymbol{m}(\delta)-\boldsymbol{m}(0)+\boldsymbol{C}(\delta)\boldsymbol{Z}-C(0)\boldsymbol{Z}\right]\right|$$

  by a similar argument.

In general, we have shown that,

$$|\triangle(\delta,\boldsymbol{Z})| \leq \sum_{i=0}^{q}\left|\boldsymbol{e}_i\left[\boldsymbol{m}(\delta)-\boldsymbol{m}(0)+\boldsymbol{C}(\delta)\boldsymbol{Z}-C(0)\boldsymbol{Z}\right]\right|,$$

so

$$\begin{aligned}
\sup_{\ell}\left|\frac{\triangle(\delta_\ell,\boldsymbol{Z})}{\delta_\ell}\right| &\leq \sum_{i=0}^{q}\sup_{\ell}\left|\frac{\boldsymbol{e}_i\left[\boldsymbol{m}(\delta_\ell)-\boldsymbol{m}(0)\right]}{\delta_\ell}+\frac{\boldsymbol{e}_i\left[\left(\boldsymbol{C}(\delta_\ell)-\boldsymbol{C}(0)\right)\boldsymbol{Z}\right]}{\delta_\ell}\right|, \\
&\leq \sum_{i=0}^{q}\sup_{\ell}\left|\frac{m(\delta_\ell)_i-m(0)_i}{\delta_\ell}\right|+\sup_{\ell}\left|\frac{C(\delta_\ell)_{i\cdot}-C(0)_{i\cdot}}{\delta_\ell}\boldsymbol{Z}\right|.
\end{aligned}$$

Define $\boldsymbol{v}$ and $\boldsymbol{V}$ such that

$$\begin{aligned}
v_i &= \sup_{\ell}\left|\frac{m(\delta_\ell)_i-m(0)_i}{\delta_\ell}\right|, \\
V_{ij} &= \sup_{\ell}\left|\frac{C(\delta_\ell)_{ij}-C(0)_{ij}}{\delta_\ell}\right|.
\end{aligned}$$

Then we have

$$\sup_{\ell}\left|\frac{\triangle(\delta_\ell,\boldsymbol{Z})}{\delta_\ell}\right| \leq \sum_{i=0}^{q}\left(v_i+\sum_{j=1}^{q}V_{ij}\left|Z_j\right|\right) =: W(\boldsymbol{Z}),$$

The conditions in Theorem 2.1 show that $\boldsymbol{m}(\delta)$ and $\boldsymbol{C}(\delta)$ are twice continuously differentiable over $[-\epsilon, \epsilon]$, and with $(\delta_l) \subseteq [-\epsilon, \epsilon]$, Lemma 2.1 assures that $v_i < \infty$, $V_{ij} < \infty \; \forall i, j$. Therefore

$$\mathbb{E}\left[W(\boldsymbol{Z})\right] = \sum_{i=0}^{q} v_i + \sum_{i=0}^{q} \sum_{j=1}^{q} V_{ij} \mathbb{E}\left|Z_j\right| < \infty.$$

Then $W(\boldsymbol{Z})$ is integrable. Also $\lim_{\ell \to \infty} \frac{\triangle(\delta_\ell, Z)}{\delta_\ell} = \frac{\partial h(\boldsymbol{X}, \boldsymbol{Z})}{\partial X_{mk}}$ exists almost surely by Lemma 2.2, where X_{mk} denotes kth dimension of $\boldsymbol{x}_m$ within $\boldsymbol{X}$. Then by Dominated Convergence Theorem (see [75]),

$$\lim_{\ell \to \infty} \mathbb{E}\left[\frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell}\right] = \mathbb{E}\left[\lim_{\ell \to \infty} \frac{\triangle(\delta_\ell, \boldsymbol{Z})}{\delta_\ell}\right]. \tag{2.14}$$

Since $\delta_\ell \searrow 0$ as $\ell \to \infty$, (2.14) becomes

$$\frac{\partial \mathbb{E} h(\boldsymbol{X}, \boldsymbol{Z})}{\partial X_{mk}} = \mathbb{E} \frac{\partial h(\boldsymbol{X}.\boldsymbol{Z})}{\partial X_{mk}}, \tag{2.15}$$

Since (2.15) applies to any i, k. $\nabla h(\boldsymbol{X}, \boldsymbol{Z})$ exists almost surely, and

$$\nabla \mathbb{E} h(\boldsymbol{X}, \boldsymbol{Z}) = \mathbb{E} \nabla h(\boldsymbol{X}, \boldsymbol{Z}).$$

□

The proof has a common approach for showing unbiasedness of an IPA estimator, by checking almost sure existence of the gradient estimator, and showing that the difference quotient is almost surely bounded by an integrable random variable (see [28, Section 1.3.1]). Indeed, it is also possible to use previously published sufficient conditions for unbiasedness of IPA estimators within the proof of Theorem 2.1, such as [28, Section 1.3.1], although this does not substantially simplify the proof, as most of the work in our proof is in verifying conditions required by this result.

Theorem 2.1 requires twice continuous differentiability of $\boldsymbol{C}(\boldsymbol{X})$, which seems difficult to verify because its analytic form is not obvious. However, following [83], which shows that mth-order differentiability of a symmetric and nonnegative definite matrix implies mth-order differentiability of the lower triangular matrix obtained from its Cholesky factorization, $\boldsymbol{L}(\boldsymbol{X})$ has the same order of differentiability as $\boldsymbol{\Sigma}^{(n)}$. In addition, (2.5) shows that $\boldsymbol{C}(\boldsymbol{X})$ has the same order of differentiability as $\boldsymbol{\Sigma}^{(n)}$, and therefore the order of differentiability for $\boldsymbol{C}(\boldsymbol{X})$ can be verified through the order of differentiability of $\boldsymbol{\Sigma}^{(n)}$, which is determined by the covariance function $k(\cdot, \cdot)$.

2.4.2 Convergence Analysis

In this section, we show our conjecture of almost sure convergence of our proposed stochastic gradient ascent algorithm. Although this algorithm assumed a search space of $\mathbb{A}^q$, we present the result where we perform the projection in (2.11) instead to a typically more flexible search space: a compact space $H = \{\boldsymbol{X} : a_i(\boldsymbol{X}) \leq 0, i = 1, \ldots, m\} \subseteq \mathbb{R}^{d \times q}$, where $a_i(\cdot)$ can be any real-valued constraint function. When $\mathbb{A}$ is compact, and can be written as $\mathbb{A} = \{\boldsymbol{x} : a'_i(\boldsymbol{x}) \leq 0, i = 1, \ldots, m'\} \subseteq \mathbb{R}^d$, where $a'_i(\cdot)$ is any real-valued constraint function, then $\mathbb{A}^q$ can be written in the form assumed for H. To do so, we write $\mathbb{A}^q = \{\boldsymbol{X} : a_{i,j}(\boldsymbol{X}) \leq 0, i = 1, \ldots, m', j = 1, \ldots, q\}$, where $a_{i,j}(\boldsymbol{X}) = a'_i(\boldsymbol{x}_j)$ and $\boldsymbol{x}_j$ is the jth point in $\boldsymbol{X}$.

The following conjecture shows that under certain conditions, including the conditions from Theorem 2.1, the stochastic gradient ascent algorithm in Section 2.3.3 converges to a stationary point almost surely. The proof will be available in a later publication of this work.

Conjecture 2.1. *Suppose the following assumptions hold,*

1. $a_i(\cdot), i = 1, \ldots, p$ *are continuously differentiable.*
2. $\epsilon_n \to 0$ *for* $n \geq 0$ *and* $\epsilon_n = 0$ *for* $n < 0$*;* $\sum_{n=1}^{\infty} \epsilon_n = \infty$ *and* $\sum_{n=0}^{\infty} \epsilon_n^2 < \infty$.
3. $\forall \boldsymbol{X} \in H$*, under the definition* (2.7)*,* $\boldsymbol{m}(\boldsymbol{X})$ *and* $\boldsymbol{C}(\boldsymbol{X})$ *are twice continuously differentiable and* $\boldsymbol{C}(\boldsymbol{X})$ *does not have duplicate rows.*

Then the sequence $\{\boldsymbol{X}_n : n = 0, 1, \ldots\}$ *generated by algorithm* (2.11) *converges to a stationary point on the q-EI surface almost surely.*

2.5 Numerical Experiments

In this section, we present numerical experiments demonstrating the usefulness of MOE-qEI. In Section 2.5.1 we compare MOE-qEI against a previously proposed heuristic from the literature, which does not attempt to solve the inner optimization problem exactly, and demonstrate that MOE-qEI is able to find better solutions to the outer optimization problem using fewer measurements. In Section 2.5.2 we show that MOE-qEI provides a nearly linear speedup over single-threaded `EGO`. In Sections 2.5.3 and 2.5.4 we compare against two previously proposed methods for solving the inner optimization problem: in Section 2.5.3 we show that MOE-qEI solves the inner optimization problem more efficiently than using closed-form evaluations of *q-EI* developed by [10] when the dimension of the problem and q is large; in Section 2.5.4 we show that the computational time of ∇q*-EI* for MOE-qEI scales better in q than the closed-form evaluation proposed by [61], and we argue that using cheap noisy evaluations in a stochastic gradient ascent framework

to solve the inner optimization problem can be more efficient than using expensive closed-form evaluations in a gradient-based optimization framework. We performed all numerical experiments using the implementation of MOE-qEI available in the open source software package "MOE". This software package is available at http://github.com/Yelp/MOE.

Although we use noise-free function evaluations, in numerical experiments, the covariance matrix $K(\cdot, \cdot)$ in (1.5) may be ill conditioned. To resolve this problem, we adopt a standard trick from Gaussian process regression and Bayesian optimization [74, Section 3.4.3]: we manually impose a small amount of noise $\sim \mathcal{N}(0, \sigma^2)$ where $\sigma^2 = 10^{-4}$ and use the noisy version of Gaussian Process regression, which is almost identical to (1.5), except that $K(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)})$ is replaced by $K(\boldsymbol{x}^{(1:n)}, \boldsymbol{x}^{(1:n)}) + \sigma^2 I_n$ where I_n is the identity matrix [74, Section 2.2].

2.5.1 Comparison Against Constant Liar Algorithm

Constant Liar is a heuristic *q-EI* algorithm proposed by [26], which uses a greedy approach to iteratively construct a batch of q points. At each iteration of this greedy approach, the heuristic uses the sequential `EGO` algorithm to find a point that maximizes the expected improvement. However, since the posterior used by `EGO` depends on points chosen for the current batch from previous iterations that have not yet been evaluated, Constant Liar imposes a dummy response (a "liar") at this point, and updates the Gaussian Process model with this "liar" value. The algorithm stops when q points are added, and reports the batch for function evaluation.

There are three variants of Constant Liar (CL), which use three different strate-

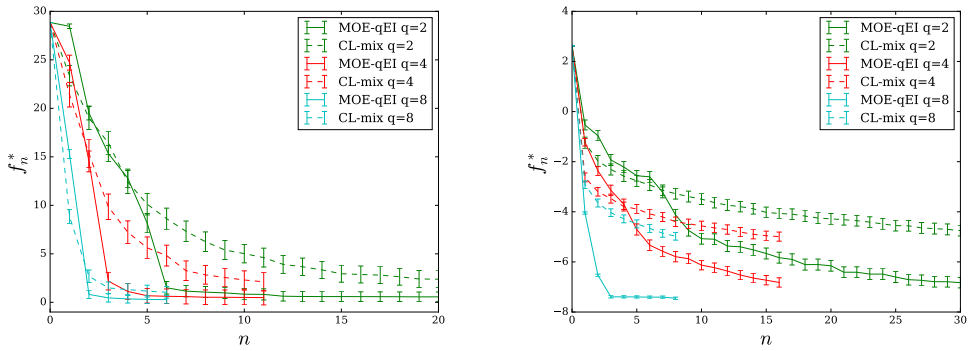
gies for choosing the liar value: CL-min sets the liar value to the minimum response observed so far; CL-max sets it to the maximum response observed so far; and CL-mix is a hybrid of CL-min and CL-max, and uses the liar value of the method that gets the highest *q-EI*. Among the three methods, CL-mix was shown by [10] to have the best overall performance, and so we compare MOE-qEI against CL-mix.

We first let both MOE-qEI and CL-mix minimize the Branin function from [14]. This is a two dimensional function and has a global minimum 0.3979. For both algorithms, we start with 15 observations in the region $\{(x_1, x_2) : -15 \leq x_1 \leq 15, -15 \leq x_2 \leq 15\}$ using Latin hypercube sampling from [62], then we construct a Gaussian Process model using the squared exponential kernel [74, Section 4.2], fit the characteristic length-scale of the kernel using the maximum likelihood approach described in [74, Section 5.4], and compute the posterior distribution as described in Section 1.2.1. Within each iteration, either algorithm (MOE-qEI or CL-mix) generates a batch of points for sampling, and the characteristic length-scale of the kernel as well as the posterior distribution of the Gaussian Process model is updated using new samples. The value of the best point f_n^* found so far at each iteration n is the performance metric in our comparison and tracks how fast an algorithm finds the global minimum. We ran both algorithms 200 times on the Branin function, and show the sample mean and 95% confidence interval for the expected value of f_n^* (averaging across the randomness in the initial stage of samples, and any residual randomness due to stochastic gradient ascent) vs. the iteration n in Figure 2.1a. The figure shows that MOE-qEI converges faster than CL-mix with equal or better solution quality in all three parallel settings, $q = 2, 4, 8$.

To show results over a large class of test problems functions, we compare the

average performance of the two algorithms, where the test objective function f is drawn at random from a Gaussian Process. We produced 200 random test objective functions from a 3-dimensional Gaussian Process, where the characteristic length-scale along each dimension is set to 5 (this number is arbitrarily chosen). To make these test problems more realistic, we did not reveal the characteristic length-scales to the Gaussian process model used by both MOE-qEI and CL-mix, instead letting the model learn the parameters from sampled points as we did when minimizing the Branin function. For each random test problem, we began with 10 observations using Latin hypercube sampling in the region $\{(x_1, x_2, x_3) : -20 \leq x_1 \leq 20, -20 \leq x_2 \leq 20, -20 \leq x_3 \leq 20\}$, and ran both MOE-qEI and CL-mix. Figure 2.1b shows the sample mean and 95% confidence interval for the expected value of f_n^* vs. the iteration n. We see that MOE-qEI significantly outperforms CL-mix on both convergence speed and solution quality.

Together, these two experiments show that, by fully solving the inner optimization problem (2.2), MOE-qEI converges faster to a solution with better quality than the heuristic method CL-mix.



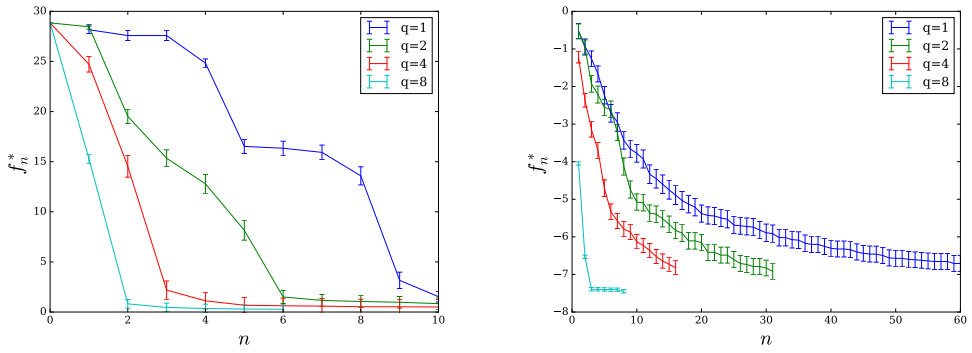
(a) Branin function

(b) Random GP-based test problems

Figure 2.1: Expected solution value, f_n^*, vs. iteration n, in the outer optimization problem, under MOE-qEI and CL-mix, for three different levels of parallelism: $q = 2$, 4, and 8 threads. We are minimizing, and so smaller function values are better. MOE-qEI converges faster with better solution quality than the heuristic method CL-mix.

2.5.2 Comparison Against `EGO`

Next, we compare MOE-qEI at different levels of parallelism against the fully sequential `EGO` algorithm, which is one-step optimal for sequential sampling. We use the same set of test functions and experiment setup as in Section 2.5.1. When $q = 1$, MOE-qEI is equivalent to `EGO` (where we use stochastic gradient ascent to maximize the expected improvement). Figure 2.2 shows the sample mean and 95% confidence interval for the expected value of f_n^* vs. n for `EGO` ($q = 1$) and MOE-qEI ($q > 1$) in minimizing Branin and random objective functions drawn from a Gaussian Process prior. Both experiments show that MOE-qEI obtains a substantial parallel speedup, and this speedup is nearly linear in q.



(a) Branin function

(b) Random GP-based test problems

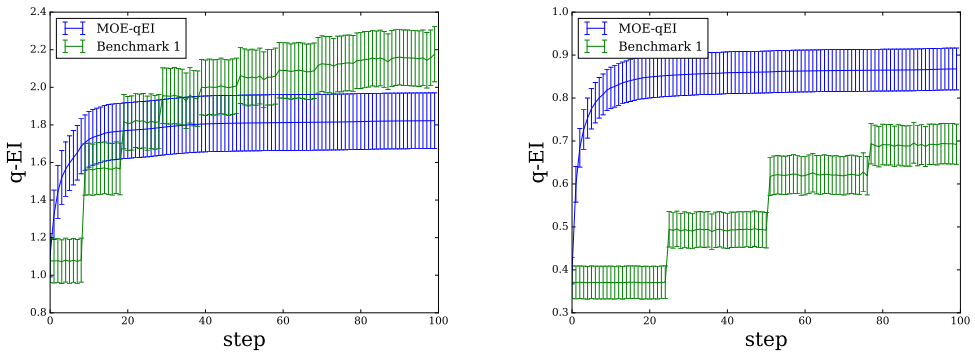
Figure 2.2: Expected solution value, f_n^* vs. iteration n, in the outer optimization problem, under `EGO` and MOE-qEI with different levels of parallelism q. MOE-qEI obtains a substantial speedup over `EGO` by evaluating in parallel, and the speedup is almost linear in q.

2.5.3 Comparison Against Closed-form Evaluation of *q-EI*

[10] provided a closed-form formula for *q-EI* and argued that it computes *q-EI* "very fast for reasonably low values of q (typically less than 10)". While having a closed-form formula is appealing, calculating this formula becomes slow and numerically challenging even with moderately large q. This is because the formula requires q^2 calls to the $q-1$ dimensional multivariate normal cdf, and computing multivariate normal cdfs with moderately large dimension is itself challenging, with state of the art methods relying on numerical integration or Monte Carlo sampling (see [23]).

While [10] did not propose using this closed-form formula to solve the inner optimization problem (2.2), one can adapt it to this purpose by using it within any derivative-free optimization method. We implemented this approach in MOE, where we use the L-BFGS [57] solver from SciPy (available at [43]) as the derivative free optimization solver. We call this approach "Benchmark 1".

We use numerical experiments to show that our Monte-Carlo based method solves the inner optimization more quickly and accurately than Benchmark 1. First we compare the speed with which Benchmark 1 and MOE-qEI can solve the inner optimization. We fix $q = 4$, and let both methods start from randomly chosen points. We conducted experiments on both test problems drawn from a 2-dimensional Gaussian process and a 6-dimensional Gaussian process. We ran both inner optimization methods 800 times and show the sample mean and 95% confidence interval for the expected solution quality (the value of *q-EI* at the current iterate) vs. iteration (step) in Figure 2.3.



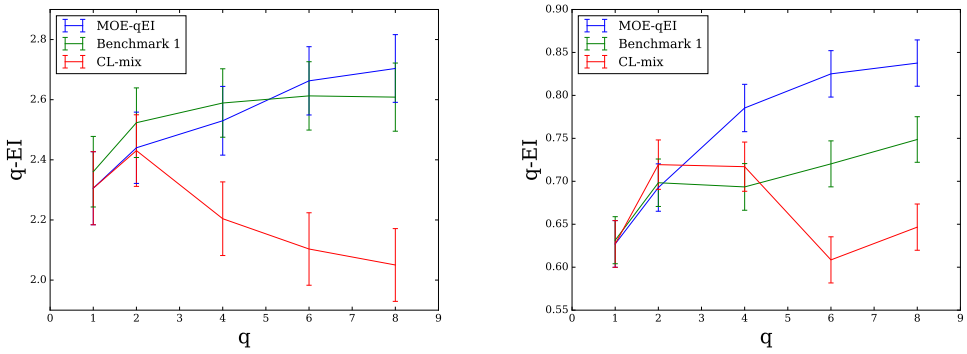
(a) 2-dim GP based problems (b) 6-dim GP based problems

Figure 2.3: Expected solution quality vs. number of steps in the inner optimization problem under MOE-qEI and Benchmark 1 (L-BFGS together with the closed form formula for q-EI from [10]). MOE-qEI's stochastic gradient ascent algorithm converges in fewer steps than L-BFGS in Benchmark 1, and each step is faster.

The figure shows that MOE-qEI's stochastic gradient ascent algorithm typically found points with better *q-EI* in fewer iterations than Benchmark 1's L-BFGS thanks to the additional gradient information. Moreover, each iteration requires substantially less time under MOE-qEI than under L-BFGS (approximately 1/10 of the time), because each step of the stochastic gradient algorithm requires only a single noisy evaluation of $\nabla q\text{-}EI$, while each step of Benchmark 1 requires mul-

tiple expensive closed-form evaluations of *q-EI* within a single step for gradient approximation and line search.

Second, we compare the quality of the solutions found to the inner optimization problem when using MOE-qEI and Benchmark 1. We use a similar experimental setup to the previous comparison, but fix the number of steps used by each method to 100. In this experimental comparison, we also include CL-mix, letting it use 100 steps in each of its greedy subproblems. We ran all three methods 250 times on 2-d problems and 575 times on 6-d problems.



(a) 2-dim GP based problems

(b) 6-dim GP based problems

Figure 2.4: Expected solution quality after 100 steps in the inner optimization problem vs. level of parallelism q, under random test problems drawn from a Gaussian process prior in 2 and 6 dimensions. We compare MOE-qEI, Benchmark 1 (L-BFGS together with the closed form formula for q-EI from [10]), and CL-mix. As q and the dimension increase, MOE-qEI finds higher quality solutions.

Figure 2.4 shows sample means and 95% confidence intervals for the expected value of the qEI of the solution to the inner optimization problem calculated by each of these methods. The figure clearly indicates that CL-mix performs the worst, especially as q increases. This behavior is expected because CL-mix is a greedy heuristic, whose suboptimality increases as more greedy steps are added with increasing q. MOE-qEI and Benchmark 1 achieved similar solution quality

on 2-d problems in the 100 iterations (though keeping in mind that iterations are more expensive under Benchmark 1), while on 6-d problems, MOE-qEI performs better when $q \geq 4$. This trend shows that as dimension and q increase and the inner optimization problem becomes more difficult, MOE-qEI tends to outperform Benchmark 1 by a larger margin.

2.5.4 Comparison Against Closed-form Evaluation of $\nabla q\text{-}EI$

A recently published book chapter [61], developed independently and in parallel to this work, proposed a method for computing $\nabla q\text{-}EI$ using a closed-form formula, and then proposed using this inside a gradient-based optimization routine to solve (2.2). This closed-form formula faces computational challenges similar in spirit to those faced by the closed-form formula for $q\text{-}EI$ proposed in [10], but even larger in magnitude. Indeed, this formula requires $O(q^4)$ calls to multivariate normal cdfs with dimension between $(q-3)$ to q. Since computing high-dimensional multivariate normal cdfs is itself challenging, this closed-form evaluation becomes extremely expensive.

MOE-qEI's Monte-Carlo based approach to evaluating $\nabla q\text{-}EI$ offers three advantages over using the closed-form formula: first, numerical experiments below suggest that computation scales better with q; second, it can be easily parallelized, with significant speedups possible through parallel computing on graphical processing units (GPUs), as is implemented within the MOE library; third, by using a small number of replications to make each iteration fast, and by using it within a stochastic gradient ascent algorithm that averages noisy gradient information

intelligently across iterations, we may more intelligently allocate effort across iterations, only spending substantial effort to estimate gradients accurately late in the process of finding a local maximum.

We first show that computation in MOE-qEI scales better with q through numerical experiments. We compare MOE-qEI with the implementation of closed-form gradient evaluation available in "DiceOptim" (see [27]), and call it "Benchmark 2". We computed $\nabla q\text{-}EI$ at 200 randomly chosen points from a 2-dimensional design space to obtain a 95% confidence interval for the average computation time. To make the gradient evaluation in MOE-qEI close to exact, we increased the number of Monte Carlo samples used in the gradient estimator to 10^7, which ensures that the variance of each component of the gradient is on the order of 10^{-10} or smaller for all q we have computed in the experiments. Figure 2.5 shows that computational time for Benchmark 2 increases quickly as q grows, but increases slowly for MOE-qEI's Monte Carlo estimator, with this Monte Carlo estimator being faster when $q \geq 4$. This difference in performance arises because gradient estimation in MOE-qEI focuses Monte Carlo effort on calculating a single high-dimensional integral, while the closed-form formula decomposes this high-dimensional integral of interest into a collection of other high-dimensional integrals that are almost equally difficult to compute, and the size of this collection grows with q.

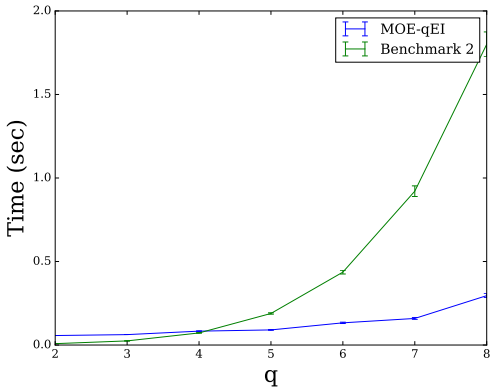


Figure 2.5: Average time to compute $\nabla q\text{-}EI$ with high precision v.s. q, comparing the gradient-based estimator from MOE-qEI using a large number of samples (10^7) with the closed-form formula from [61]. The stochastic gradient estimator in MOE-qEI scales better in q and is faster when $q \geq 4$

When we use this Monte Carlo estimator within MOE-qEI, we may obtain additional speed improvements by not using as many Monte Carlo samples. This is possible because stochastic gradient ascent is able to handle noisy gradient estimation. Using fewer Monte Carlo samples in each iteration increases efficiency by only putting effort toward estimating the gradient precisely when we are close to the stationary point, which stochastic gradient ascent performs automatically through its decreasing stepsize sequence. Indeed, we used 10^6 Monte Carlo samples for gradient estimation in all the other experiments we have conducted, which provides an order of magnitude speedup in each iteration, without dramatically increasing the number of iterations required.

2.6 Summary

We proposed an efficient method based on stochastic approximation for implementing a conceptual parallel Bayesian global optimization algorithm proposed by [26].

To accomplish this, we used infinitesimal perturbation analysis (IPA) to construct a stochastic gradient estimator and showed that this estimator is unbiased. We also provided convergence analysis of the stochastic gradient ascent algorithm with the constructed gradient estimator. Through numerical experiments, we demonstrate that our method outperforms the existing state-of-the-art approximation methods.

CHAPTER 3

PEPTIDE OPTIMIZATION USING DISCRETE BAYESIAN OPTIMIZATION

3.1 Introduction

In this chapter, we propose a discrete Bayesian optimization method to address a specific instance of a problem common in medicine, materials science, chemistry, and other physical sciences. In this class of problems, we wish to find a molecule with a particular biological or chemical property. While we have historical data on related molecules or related properties that can support predictive modeling, the only way that we can determine with certainty whether a particular molecule has the property of interest (in colloquial language, "is a hit") is to synthesize and test it in a time-consuming laboratory experiment. Success is hampered by the enormous search space, and the limits imposed by experimental cost on the number of molecules that can be tested. Within this context, we wish to use mathematical methods to recommend which experiments to perform, so as to reliably find a hit within a given experimental budget. Common examples include drug discovery, in which we wish to find a molecule that can be effective in treating a particular disease [66], and materials discovery [72, 17], in which we wish to find a molecule that has a particular physical or chemical property.

The instance of this problem that we consider in this chapter comes from biochemistry: we wish to find a peptide that (1) is as short as possible; (2) is a substrate for protein-modifying enzymes, phosphopantetheinyltransferase (PPTase) and ACP hydrolase (AcpH); and (3) is orthogonal with respect to two different classes of PPTases, the Sfp and AcpS-type, which means the peptide is a sub-

strate for Sfp-type PPTases, and is not a substrate for AcpS-type PPTases, and vice versa. These protein-modifying enzymes can be used to attach and detach arbitrary molecules to their substrates. In particular, as shown in Figure 3.1, we can use PPTase to attach a fluorescent "tag" or "label" to peptides that are substrates for this PPTase, allowing us to track the peptide, starting from a time of our choosing. If the peptides are also substrates for AcpH, we can then use AcpH to remove the fluorescent tag at a time of our choosing, Moreover, if the peptides are orthogonal with respect to Sfp and AcpS-type PPTases, we gain even finer control over which peptides get labeled using their feature of selectivity. If this peptide were short, then it could be inserted into a protein or peptide-based catalytic complex of our choosing without disrupting its activity, allowing monitoring of biological and catalytic systems. This monitoring tool would have a number of applications in medicine, biology, and materials science [51].

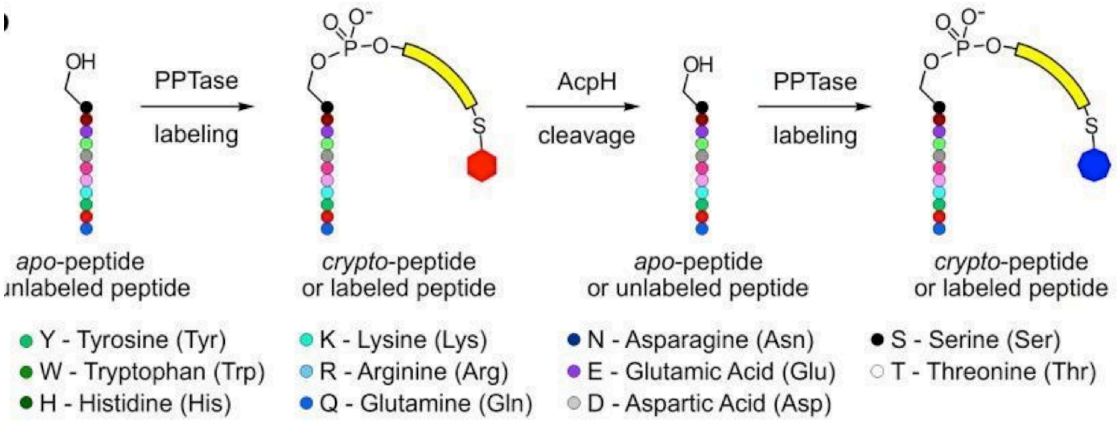


Figure 3.1: Illustration of the reversible labeling process: PPTase modifies the peptide at a specific Serine residue by addition of a fluorescent molecule, i.e., the labeling process; then AcpH removes this modification, i.e., the unlabeling process. The process can be repeated for many times.

Finding such peptides is challenging, because the number of peptides for consideration is huge, and only a tiny fraction of them satisfy all aforementioned requirements (i.e., are "hits"). In fact, prior to this study, there was only one pep-

tide discovered through Phage Display [100, 102] that satisfies all criteria posed above. We could make a rough estimate: a peptide is a string of amino acids, of which 20 occur in nature, so there are 20^n distinct peptides of length n; the typical length of peptides for consideration is around 15 amino acids, and therefore the total number of peptides is $20^{15} \approx 3.3 \times 10^{19}$. Of these many choices, our collaborators from biochemistry estimate that less than 1 in 10^5 is a substrate for any of the enzymes of interest, and the chance of finding a peptide that is a substrate for multiple types of enzymes is even smaller. We can test peptides in batches of size approximately 600, and given the experimental budget, we could at most test five batches, which amounts to 3000 peptides. If we were to test 3000 peptides of length 15 uniformly at random, we have at most $p = 1 - (1 - 10^{-5})^{3000} \approx 3\%$ chance to find at least one hit. This is discouraging considering that this effort would take years and tens of thousands of dollars. Even if we leverage machine learning to guide experimentation, the enormous size of the space of peptides to search, and the small size of the training set, both pose a formidable challenge to obtaining accurate predictions of biological activity over the search space.

Contributions. To address these challenges, we formulate this problem as a discrete Bayesian optimization problem. Since the Gaussian Process regression model that we reviewed in Section 1.2.1 is for modeling continuous functions, and is not for classification tasks, we develop a Bayesian classification model, that combines Naïve Bayes [59], a prior distribution encoding knowledge from domain experts, and training data from naturally occurring and longer peptide substrates, to predict whether a peptide is a hit. We then use the joint probability distributed over unobserved labels ("hit" or "miss") of all peptides created by this model within a combinatorial optimization framework to find a *set* of peptides to test that,

when taken together, are most likely to produce a hit. We provide a theoretical performance guarantee on the quality of the resulting set of peptides to test, and using simulation, we show that this combination of prediction and optimization provides robust results that are better than a more naive use of the predictive model that considers only marginal probabilities, and a benchmark method of truncating and mutating known hits at random. We then describe laboratory results in which this method was used to discover peptide substrates that are shorter than the shortest previously known: the ybbR sequence [102].

While the methods we develop were designed to address a particular problem arising in biochemistry, the formal model we propose is general, and can be used whenever we have a large collection of exemplars with expensive-to-obtain labels, and we wish to choose exemplars to evaluate so as to find one with both a positive label and a small secondary cost objective, within a limited budget.

Related Work. The method proposed in this chapter is related to a number of recently developed methods for optimal search, all of which aim to effectively collect information so as to make the best decisions under uncertainty. In this setting, they need to trade off the reward by sampling (i.e. exploitation) and the cost by acquiring this information (i.e. exploration). There are a number of methods in the similar setting proposed in the active learning research community in the past a few years. For example, in adaptive information filtering, a new measure of model quality that quantitatively manage trade-off between expoitation and exploration [101] was proposed. In information retrieval, there are a few papers about batch mode (parallel sampling) active learning in various contexts [9, 35, 34], although they only aim to maximize information gain (i.e. pure exploration).

Outline of this Chapter In section 3.2, we provide a formal model of our problem. In section 3.3, we state the Bayesian classification model used for the problem. In section 3.4, we formulate our sampling strategy, prove its theoretical performance guarantee, and summarize it in pseudo code. In section 3.6, we use real data to validate our statistical model, compare the performance of our method with other benchmark methods, and show the real experimental outcome of the biochemical application.

3.2 Problem Formulation

We now formulate the motivating biochemical application as a discrete Bayesian optimization problem. Let E be a generic space of exemplars. In our application, E is the space of peptides. Each element $x \in E$ has an unknown binary label $y(x) = \{0, 1\}$. In our application, $y(x)$ is 1 if x is a hit, i.e., is a substrate for enzymes of interest, and otherwise 0. A known deterministic function $f(x)$ measures the cost or disutility associated with x. In our application, $f(x)$ is the length of the peptide, as longer peptides interfere more with the system being monitored. Our goal is to perform experiments so as to find x with a positive label and a cost $f(x)$ that is as small as possible.

To obtain labels of exemplars, we can do experiments in batches, which evaluate the labels for a subset $S \subseteq E$. The cardinality of S is constrained by some K (i.e. $|S| \leq K$), and this is typically determined by the total number of peptides the experimental equipment can handle in a batch. We measure the quality of S by

$$f^*(S) = \min_{x \in S : y(x) = 1} f(x), \tag{3.1}$$

where we assume $\min \emptyset = \infty$. $f^*(S)$ measures the smallest cost function for positive

labeled elements in the set S, and in our application, is the shortest length of hits in the set of peptides to test.

Let b be a target value, then we wish to find at least one x such that x is a hit and $f(x)$ is smaller than b. Since we do experiments in batches, we want to find such a batch S, that $f^*(S)$ is smaller than b. In our application, we let b be the length of the shortest previously known hit. $f^*(S)$ is unknown because we do not know the label $y(x)$ for $x \in S$ before the experiment. We assume a joint probability distribution over $y(x) : x \in E$, which is determined by the underlying statistical model for a particular problem, with the model used in our application described in Section 3.3. We pursue a Bayesian optimization approach, and consider probability of improvement and expected improvement, defined as follows:

$$\begin{aligned} &\text{probability of improvement:} && \texttt{PI}(S) = \mathbb{P}(f^*(S) < b) \\ &\text{expected improvement:} && \texttt{EI}(S) = \mathbb{E}[(b - f^*(S))^+] \end{aligned} \tag{3.2}$$

Given a constraint on the cardinality of S, we wish to find S that maximizes one of these two criteria. Let $g(S)$ be either $\texttt{PI}(S)$ or $\texttt{EI}(S)$, and then our sampling strategy is to find the set S to sample by solving, or approximately solving,

$$\max_{S \subseteq E : |S| \leq K} g(S). \tag{3.3}$$

3.3 Bayesian Naïve Bayes Model

For an x where its label is unknown, we build a statistical model to predict the corresponding $y(x)$. In the Bayesian optimization framework, as discussed in Section 1.2.1, we need a Bayesian model that predicts the posterior distribution over $y(x)$, so as to compute the subsequent acquisition function. To this goal, we develop a Bayesian classification model that combines Naïve Bayes and a prior

distribution encoding knowledge from domain experts. Let $X = (X_1, \ldots, X_L)$ be a feature vector and Y be its label. Using Bayes' Rule, we have:

$$\mathbb{P}(Y = y|X = x) = \frac{\mathbb{P}(X = x|Y = y)\mathbb{P}(Y = y)}{\mathbb{P}(X = x)} = \frac{\mathbb{P}(X = x|Y = y)\mathbb{P}(Y = y)}{\sum_{y'} \mathbb{P}(X = x|Y = y')\mathbb{P}(Y = y')}.$$

The Naïve Bayes classifier assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature, given the class variable, i.e.

$$\mathbb{P}(Y = y|X = x) = \frac{\prod_{j=1}^{L} \mathbb{P}(X_j = x_j|Y = y)\mathbb{P}(Y = y)}{\sum_{y'} \prod_{j=1}^{L} \mathbb{P}(X_j = x_j|Y = y)\mathbb{P}(Y = y)}.$$

In the context of our application, Y is a random variable that is either "1" or "0", depicts whether a peptide is a "hit" or "miss"; X is feature vector that encodes any peptide of interest, which is designed in the following: since a peptide is a sequence of amino acids, we use $A_j \in \{1, \ldots, 20\}$ to indicate the type of amino acid in jth position of the sequence, where $j = 1, \ldots, L$; we further partition the 20 different types of amino acids into K groups ($K \leq 20$) according to their biochemical similarities, then jth feature is simply indicating which group the amino acid in the jth position of the sequence belongs to, denoted as x_j.

We let $\theta_{y,j}(k) = \mathbb{P}(X_i = k|Y(X) = y)$, where $j = 1, \ldots, L$, $k = 1, \ldots, K$ and $y \in \{0, 1\}$. Then given a prior distribution $\mathbb{P}(Y(x) = y)$, $y \in \{0, 1\}$, the Bayesian inference for any peptide x is

$$\mathbb{P}\left(Y(x) = 1|\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, x\right) = \frac{\mathbb{P}(Y(x) = 1) \prod_j \theta_{1,j}(x_j)}{\left[\mathbb{P}(Y(x) = 1) \prod_j \theta_{1,j}(x_j)\right] + \left[\mathbb{P}(Y(x) = 0) \prod_j \theta_{0,j}(x_j)\right]}, \tag{3.4}$$

where $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ are the matrix representations of $\theta_{0,j}(k)$ and $\theta_{1,j}(k)$.

We train the model parameters $\theta_{y,j}(k)$ using Bayesian inference. We assume the prior distribution over $\theta_{y,j}(\cdot)$ is Dirichlet$(\alpha_{y,j}(1), \ldots, \alpha_{y,j}(K))$, and we can encode prior knowledge from domain experts into the choices of the prior parameters $\alpha_{y,j}(k)$. In this biochemical application, the enzymes can only attach/detach "label" to/from a special position in the peptide, and we know that those positions further away from this special position will have less influence on whether the peptide is a hit or not. To encode this information into the prior, we increase the value of $\alpha_{y,j}(k)$ and make it uniform over k as j moves further away from the known position, and this encodes the belief that the positions far away from the special position is less sensitive about which amino acid to appear over that position. From Bayesian statistics, we know that the posterior is Dirichlet$(\alpha_{y,j}(1) + N_{y,j}(1), \ldots, \alpha_{y,j}(K) + N_{y,j}(K))$, where $N_{y,j}(k)$ is the number of the peptides in the training set with $Y(x) = y$, and $x_j = k$.

3.4 The Sampling Strategy

In Bayesian optimization, we use an acquisition function as our sampling criterion, and sample the point that maximizes over this acquisition function. We employ the similar approach here, and use (3.2) as our sampling criteria. However, solving (3.3) is rather difficult: it is a combinatorial optimization problem with a huge feasible space. This problem is, in fact, hard for any known algorithm to solves it quickly. We propose a greedy algorithm to solve it approximately, that is, begin with the empty set $S = \emptyset$, and iteratively find element e such that

$$\arg\max_{e \in E \setminus S} g(S \cup \{e\}), \tag{3.5}$$

and incorporate it into S. This process stops until $|S| = K$ for some chosen K. This approach reduces the size of search space dramatically from $|E|^{|S|}$ down to $|E| \times |S|$. Later we will show that this approximation does not undermine the solution quality significantly, in fact, when the objective function g is either `PI` or `EI`, the proposed greedy algorithm is guaranteed to be near-optimal with a lower bound. Additionally, we show that using the model described in section 3.3, we can formulate (3.5) as Mixed-Integer Nonlinear Programming (MINLP), and solve it efficiently using off-the-shelf MINLP solvers.

3.4.1 Performance Guarantee for the Greedy Algorithm

In this section, we show that when the objective function g in (3.5) is either `PI` or `EI`, the proposed greedy algorithm has a near-optimal performance guarantee. The result is as follows:

Theorem 3.1. *If objective function is probability of improvement (i.e* $\texttt{PI}(S)$*) or expected improvement (i.e* $\texttt{EI}(S)$*), the greedy algorithm is guaranteed to achieve a factor of* $(1 - 1/e)(\approx 63\%)$ *of the optimal solution.*

We prove the theorem using Lemma 3.1, 3.2, and 3.3. Lemma 3.1 is a result from [67], which analyzes the performance lower bound of greedy heuristic in combinatorial optimization for a certain class of objective functions. Lemma 3.2 and 3.3 show that `PI` and `EI` are the objective functions that satisfy conditions in Lemma 3.1, and therefore greedy algorithm has a performance lower bound.

Lemma 3.1. *If* $F(S)$ *is submodular, nondecreasing and* $F(\emptyset) = 0$*, the greedy heuristic always produces a solution whose value is at least* $1 - [(K-1)/K]^K$ *times*

the optimal value, where $|S| \leq K$. This bound can be achieved for each K and has a limiting value of $1 - 1/e$, where e is the base of the natural logarithm.

Lemma 3.2. *Probability of improvement $\texttt{PI}(S)$ is submodular, nondecreasing and $\texttt{PI}(\emptyset) = 0$.*

Lemma 3.3. *Expected improvement $\texttt{EI}(S)$ is submodular, nondecreasing and $\texttt{EI}(\emptyset) = 0$.*

Proof of Theorem 3.1. From Lemma 3.2 and 3.3, we know that PI and EI are submodular, nondecreasing and their measure of the empty set is 0. From Lemma 3.1, we conclude that if the objective function g in (3.5) is PI or EI, the greedy algorithm is guaranteed to achieve a factor of $(1 - 1/e)$ of the optimal value. □

Proof of Lemma 3.2. First we show $\texttt{PI}(\emptyset) = 0$.

$$\texttt{PI}(\emptyset) = \mathbb{P}(f^*(\emptyset) < b) = \mathbb{P}(\infty < b) = 0.$$

To show $\texttt{PI}(S)$ is nondecreasing, let $A \subseteq B \subseteq E$ where E is a finite set, then

$$\begin{aligned}
\texttt{PI}(B) &= \mathbb{P}(f^*(B) < b) \\
&= \mathbb{P}(f^*(B) < b | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) + \mathbb{P}(f^*(B) < b | f^*(A) < b)\mathbb{P}(f^*(A) < b) \\
&= \mathbb{P}(f^*(B) < b | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) + \mathbb{P}(f^*(A) < b) \\
&\geq \mathbb{P}(f^*(A) < b) \\
&= \texttt{PI}(A)
\end{aligned}$$

Lastly, we want to show $\texttt{PI}(S)$ is submodular. For $e \in E \backslash B$,

$$
\begin{aligned}
&\texttt{PI}(A \cup \{e\}) - \texttt{PI}(A) \\
&= \mathbb{P}(f^*(A \cup \{e\}) < b) - \mathbb{P}(f^*(A) < b) \\
&= \mathbb{P}(f^*(A \cup \{e\}) < b | f^*(A) < b)\mathbb{P}(f^*(A) < b) \\
&\quad + \mathbb{P}(f^*(A \cup \{e\}) < b | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) - \mathbb{P}(f^*(A) < b) \\
&= \mathbb{P}(f^*(A) < b) + \mathbb{P}(f^*(A \cup \{e\}) < b | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) - \mathbb{P}(f^*(A) < b) \\
&= \mathbb{P}(f^*(A \cup \{e\}) < b | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) \\
&= \mathbb{P}(f(e) < b, y(e) = 1 | f^*(A) \geq b)\mathbb{P}(f^*(A) \geq b) \\
&= \mathbb{P}(f(e) < b, y(e) = 1, f^*(A) \geq b)
\end{aligned}
$$

Using similar argument,

$$
\begin{aligned}
&\texttt{PI}(B \cup \{e\}) - \texttt{PI}(B) \\
&= \mathbb{P}(f(e) < b, y(e) = 1, f^*(B) \geq b) \\
&= \mathbb{P}(f(e) < b, y(e) = 1, f^*(A) \geq b, f^*(B \backslash A) \geq b)
\end{aligned}
$$

Therefore, $\texttt{PI}(A \cup \{e\}) - \texttt{PI}(A) \geq \texttt{PI}(B \cup \{e\}) - \texttt{PI}(B)$, thus $\texttt{PI}(S)$ is submodular.

□

Proof of Lemma 3.3. First we show that $\texttt{EI}(\emptyset) = 0$.

$$\texttt{EI}(\emptyset) = \mathbb{E}[(b - f^*(\emptyset))^+] = \mathbb{E}[0] = 0.$$

To show $\texttt{EI}(S)$ is nondecreasing, let $A \subseteq B \subseteq E$ where E is a finite set. Since $f^*(B) \leq f^*(A)$, $b - f^*(B) \geq b - f^*(A)$, and $(b - f^*(B))^+ \geq (b - f^*(A))^+$, therefore, $\mathbb{E}[(b - f^*(B))^+] \geq \mathbb{E}[(b - f^*(A))^+]$.

Lastly, we want to show $\texttt{PI}(S)$ is submodular. For $e \in E \backslash B$, consider $\mathbb{E}[(b -$

$f^*(A \cup \{e\}))^+] - \mathbb{E}[(b - f^*(A))^+]$. We can write

$$(b - f^*(A \cup \{e\}))^+ = \begin{cases} (b - f^*(A))^+ & \text{if } y(e) = 0 \\ (b - \min\{f(e), f^*(A)\})^+ & \text{if } y(e) = 1 \end{cases}$$

Then

$$\begin{aligned}
&\mathbb{E}[(b - f^*(A \cup \{e\}))^+] - \mathbb{E}[(b - f^*(A))^+] \\
&= \mathbb{P}(y(e) = 1)\mathbb{E}[(b - \min\{f(e), f^*(A)\})^+ - (b - f^*(A))^+ | y(e) = 1] \\
&= \mathbb{P}(y(e) = 1)\mathbb{P}(f(e) < f^*(A) | y(e) = 1)\mathbb{E}[(b - e)^+ - (b - f^*(A))^+ | y(e) = 1, f(e) < f^*(A)] \\
&= \mathbb{E}[\mathbb{1}_{y(e)=1, f(e)<f^*(A)}((b - e)^+ - (b - f^*(A))^+)]
\end{aligned}$$

Since $f^*(A) \geq f^*(B)$, $\mathbb{1}_{y(e)=1,f(e)<f^*(A)}((b - e)^+ - (b - f^*(A))^+)) \geq \mathbb{1}_{y(e)=1,f(e)<f^*(B)}((b - e)^+ - (b - f^*(B))^+))$, thus

$$\texttt{EI}(A \cup \{e\}) - \texttt{EI}(A) \geq \texttt{EI}(B \cup \{e\}) - \texttt{EI}(B)$$

$\texttt{EI}(S)$ is submodular. □

3.4.2 Efficient Formulation for Probability of Improvement

When the acquisition function is probability of improvement, we write (3.5) as

$$\arg\max_{e \in E \setminus S} \mathbb{P}\left(f^*(S \cup \{e\})\right). \tag{3.6}$$

In Proposition 3.1, we show that (3.6) can be written into the form that the objective function is easy to compute. In fact, this is the general form of greedy algorithm for optimization over probability of improvement, and can be used with any underlying classification model. In addition, under the Naïve Bayes model discussed in Section 3.3, we can further write (3.6) as a MINLP, which can be solved efficiently by any off-the-shelf MINLP solver.

Proposition 3.1. *If the objective function g is* `PI`*, we can write* (3.5) *as*

$$\underset{e\in E\setminus S, f(e)<b}{\arg\max}\ \mathbb{P}(y(e)=1|y(x)=0,\forall x\in S). \tag{3.7}$$

Proof of proposition 3.1.

$$\begin{aligned}&\texttt{PI}(S\cup\{e\}) = \mathbb{P}(f^*(S\cup\{e\})<b)\\ &= \mathbb{P}(f^*(S)<b)+\mathbb{P}(f^*(S)\geq b)\mathbb{P}(f(e)<b, y(e)=1|f^*(S)\geq b),\end{aligned}$$

so (3.5) becomes

$$\max_{e\in E\setminus S}\texttt{PI}(S\cup\{e\}) = \max_{e\in E\setminus S}\mathbb{P}(f(e)<b, y(e)=1|f^*(S)\geq b). \tag{3.8}$$

Note that when $f(e)\geq b$, $\mathbb{P}(f(e)<b, y(e)=1|f^*(S)\geq b)=0$, thus our algorithm will always propose e such that $f(e)<b$. Therefore, it is reasonable to assume that $f(x)<b$ for $\forall x\in S$, and $f^*(S)\geq b$ is equivalent to $y(x)=0$ for $\forall x\in S$. Now we can write (3.8) as

$$\max_{e\in E\setminus S, f(e)<b}\mathbb{P}(y(e)=1\mid y(x)=0,\forall x\in S).$$

□

We can formulate (3.7) as a MINLP under the model described in Section 3.3. First we write (3.4) as

$$\mathbb{P}\left(Y(x)=1|\theta\right) = \frac{\prod_j \eta_j(x_j)}{\prod_j \eta_j(x_j)+\frac{\mathbb{P}(Y(x)=0)}{\mathbb{P}(Y(x)=1)}}, \tag{3.9}$$

where

$$\eta_j(x_j) = \frac{\theta_{1,j}(x_j)}{\theta_{0,j}(x_j)}\ \text{ for } \forall j\in\{1,\ldots,L\}.$$

Plug (3.9) into (3.7), and we get

$$\underset{e\in E\setminus S, f(e)<b}{\arg\max}\ \frac{\prod_j \eta_j(e_j)}{\prod_j \eta_j(e_j)+\frac{\mathbb{P}(Y(e)=0)}{\mathbb{P}(Y(e)=1)}}, \tag{3.10}$$

where

$$\eta_j(e_j) = \frac{\mathbb{P}(e_j|Y(e)=1, Y(x)=0, \forall x \in S)}{\mathbb{P}(e_j|Y(e)=0, Y(x)=0, \forall x \in S)}.$$

We write (3.10) in the form of a MINLP,

$$\begin{aligned} \max \quad & \frac{\prod_j \Sigma_k x_j(k)\eta_j(k)}{\prod_j \Sigma_k x_j(k)\eta_j(k) + \frac{\mathbb{P}(Y(x)=0)}{\mathbb{P}(Y(x)=1)}} \\ \text{s.t} \quad & k \in \{1, \ldots, K\} \\ & x_j(k) \in \{0, 1\} \\ & \Sigma_k x_j(k) = 1, \end{aligned} \tag{3.11}$$

where

$$x_j(k) = \begin{cases} 1 & \text{if } e_j = k \\ 0 & \text{else.} \end{cases}$$

We summarize the algorithm in the following:

Algorithm 3.1. *(Probability of Improvement)*

Require: *Inputs M, J, K, data set D and prior distribution of* $\theta_y \sim$ *Dirichlet*$(\boldsymbol{\alpha}_y), y \in \{1, 0\}$

1: $S \leftarrow \emptyset$

2: *Calculate posterior distribution of* $\theta_1 \sim$ *Dirichlet*$(\boldsymbol{\alpha}_1|\{x|x \in D, y(x) = 1\})$.

3: ***for*** $m = 1$ *to* M ***do***

4: $\quad COUNT \leftarrow 0$

5: $\quad$ *Calculate posterior distribution of* $\theta_0 \sim$ *Dirichlet*$(\boldsymbol{\alpha}_0|\{x|x \in D, y(x) = 0\} \cup S)$.

6: $\quad$ ***loop***

7: $\quad\quad$ *Sample* θ_1 *from Dirichlet*$(\boldsymbol{\alpha}_1|\{x|x \in D, y(x) = 1\})$ *and* θ_0 *from Dirichlet*$(\boldsymbol{\alpha}_0|\{x|x \in D, y(x) = 0\} \cup S)$.

8: $\quad\quad \eta \leftarrow \frac{\theta_1}{\theta_0}$

9: $\quad\quad$ *Solve MINLP in equation* (3.11) *to find* x.

10: $COUNT \leftarrow COUNT + x.$

11: ***end loop***

12: ***for*** $j = 1$ *to* J ***do***

13: $e_j \leftarrow \underset{k \in \{1,\ldots,K\}}{\arg\max}\ COUNT_{kj}$

14: ***end for***

15: $S \leftarrow (S, e)$

16: ***end for***

3.4.3 Efficient Formulation for Expected Improvement

If we use expected improvement as acquisition function instead, similar to probability of improvement, we first want to simplify the formulation, and the result is shown in Proposition 3.2.

Proposition 3.2. *If the objective function g is* `EI`*, we can write* (3.5) *as*

$$\underset{e \in E \setminus S}{\arg\max}\, c_0 \mathbb{P}_0(e)(b - f(e))^+ + \sum_{i=1}^{|S|} c_i \mathbb{P}_i(e)(f(x_i) - f(e))^+, \tag{3.12}$$

where

$$\mathbb{P}_0(e) = \mathbb{P}(y(e) = 1 | y(x) = 0, \forall x \in S),$$

$$\mathbb{P}_i(e) = \mathbb{P}(y(e) = 1 | y(x_i) = 1, y(x_j) = 0, \forall j < i, x_i, x_j \in S),$$

and $c_i (i = 0, \ldots, |S|)$ are known coefficients.

Proof of proposition 3.2. Since choosing e such that $f(e) \geq b$ has no contribution to the objective function, by using similar argument as dealing with probability of improvement, we argue that $f(x) < b$ for $\forall x \in S$. Thus

$$f^*(S) \begin{cases} = \infty & \text{if } y(x) = 0 \text{ for } \forall x \in S, \\ < b & \text{else.} \end{cases}$$

Now objective function we want to maximize becomes

$$\mathbb{E}\left[(b - f^*(S \cup \{e\}))^+\right]$$
$$= \mathbb{E}[(b - f(e))^+ \mathbb{1}_{f^*(S)=\infty, y(e)=1}] + \mathbb{E}[(b - f^*(S \cup \{e\}))^+ \mathbb{1}_{f^*(S)<b}]$$
$$= \mathbb{E}[(b - f(e))^+ \mathbb{1}_{f^*(S)=\infty, y(e)=1}] + \mathbb{E}[(b - f^*(S)) \mathbb{1}_{f^*(S)<b}] + \mathbb{E}[(f^*(S) - f(e)) \mathbb{1}_{y(e)=1, f(e)<f^*(S)<b}].$$

so

$$\max_{e \in E \backslash S} \mathbb{E}\left[(b - f^*(S \cup \{e\}))^+\right],$$
$$= \max_{e \in E \backslash S, f(e)<b} \mathbb{E}[(b - f(e)) \mathbb{1}_{f^*(S)=\infty, y(e)=1}] + \mathbb{E}[(f^*(S) - f(e)) \mathbb{1}_{y(e)=1, f(e)<f^*(S)<b}]. \tag{3.13}$$

For $e \in E \backslash S, f(e) < b$,

$$\mathbb{E}[(b - f(e)) \mathbb{1}_{f^*(S)=\infty, y(e)=1}] = \mathbb{P}(y(e) = 1, y(x) = 0, \forall x \in S)(b - f(e)), \tag{3.14}$$

$$\mathbb{E}[(f^*(S) - f(e)) \mathbb{1}_{y(e)=1, f(e)<f^*(S)<b}],$$
$$= \mathbb{E}[\mathbb{E}[(f^*(S) - f(e)) \mathbb{1}_{y(e)=1, f(e)<f^*(S)<b}] | f^*(S) = l],$$
$$= \sum_{l \in L, f(e)<l} \mathbb{P}(y(e) = 1 | f^*(S) = l)(l - f(e)) \mathbb{P}(f^*(S) = l),$$

where $L = \{f(x) : x \in S\}$. If we rank elements in S such that $f(x_i) \leq f(x_j), \forall i < j, x_i, x_j \in S$, we can write equation above as

$$\sum_{i=1}^{|S|} \mathbb{P}(y(e) = 1, y(x_i) = 1, y(x_j) = 0, \forall j < i, x_i, x_j \in S)(f(x_i) - f(e))^+. \tag{3.15}$$

Substitute (3.14) (3.15) into (3.13), and note that $\mathbb{P}(y(e) = 1, \mathcal{F}(x_1, \ldots, x_{|S|}) \propto \mathbb{P}(y(e) = 1 | \mathcal{F}(x_1, \ldots, x_{|S|})$ with known coefficient given S, we get (3.12) in Proposition 2. □

We can re-formulate (3.12) as a MINLP:

$$\begin{aligned} \max \quad & \Sigma_{i=0}^{|S|} c_i \frac{\prod_j \Sigma_k x_j(k)\eta_j^i(k)}{\prod_j \Sigma_k x_j(k)\eta_j^i(k) + \frac{\mathbb{P}(Y(x)=0)}{\mathbb{P}(Y(x)=1)}} (f_i - f(e))^+ \\ \text{s.t} \quad & k \in \{1, \ldots, K\} \\ & x_j(k) \in \{0,1\} \\ & \Sigma_k x_j(k) = 1, \end{aligned} \tag{3.16}$$

where

$$x_j(k) = \begin{cases} 1 & \text{if } e_j = k \\ 0 & \text{else,} \end{cases}$$

$$f_i = \begin{cases} b & \text{if } i = 0 \\ f(p^i) & \text{else,} \end{cases}$$

and c_i's are known coefficients. We summarize the algorithm as follows:

Algorithm 3.2. *(Expected Improvement)*

Require: *Inputs M, J, K, data set D and prior distribution of* $\theta_y \sim$ *Dirichlet*$(\boldsymbol{\alpha}_y), y \in \{1,0\}$

1: $S \leftarrow \emptyset$

2: ***for*** $m = 1$ *to* M ***do***

3: $COUNT \leftarrow 0$

4: ***if*** S *is not empty* ***then***

5: *Sort elements in* S *as* $\{p^1, \ldots, p^{|S|}\}$ *such that* $f(p^i) \leq f(p^j), \forall i < j$.

6: ***end if***

7: *Calculate posterior distribution of* $\theta_1^0 \sim$ *Dirichlet*$(\boldsymbol{\alpha}_1 | \{x | x \in D, y(x) = 1\})$ *and* $\theta_0^0 \sim$ *Dirichlet*$(\boldsymbol{\alpha}_0 | \{x | x \in D, y(x) = 0\} \cup S)$.

8: ***for*** $i = 1$ *to* $|S|$ ***do***

9: *Calculate posterior distribution of* $\theta_1^i \sim$ *Dirichlet*$(\boldsymbol{\alpha}_1 | \{x | x \in D, y(x) = 1\} \cup \{p^i\})$ *and* $\theta_0^i \sim$ *Dirichlet*$(\boldsymbol{\alpha}_0 | \{x | x \in D, y(x) = 0\} \cup \{p^j | j < i\})$.

10: ***end for***

11: ***loop***

12: *Sample* $\theta_1^{i=0:|S|}$ *and* $\theta_0^{i=0:|S|}$ *from posterior distribution.*

13: $\eta^{i=0:|S|} \leftarrow \frac{\theta_1^{i=0:|S|}}{\theta_0^{i=0:|S|}}$

14: *Solve MINLP in equation* (**??**) *to find* x.

15: $COUNT \leftarrow COUNT + x$.

16: ***end loop***

17: ***for*** $j = 1$ *to* J ***do***

18: $e_j \leftarrow \underset{k \in \{1,\ldots,K\}}{\arg\max}\ COUNT_{kj}$

19: ***end for***

20: $S \leftarrow (S, e)$

21: ***end for***

Remark 3.1. *Comparing the objective functions in* (3.7) *and* (3.12)*, you will notice that computation of* `PI` *is a lot cheaper than computing* `EI`*, and thus solving optimization problem with* `PI` *is a lot easier than solving* `EI`*, which you can observe by comparing* (3.11) *and* (3.16)*. However, the trade-off is that,* `PI` *only finds hits with length shorter than some threshold, whereas* `EI` *aims to optimize over length directly and will find hits with minimal lengths. The characteristics of the two acquisition functions were discussed earlier in Section 1.2.2 and 1.2.2. Both methods have their own advantages in achieving the goal.*

3.5 Numerical Experiments

In this section, we use numerical study to validate our statistical model and sampling strategy.

Model Validation In Section 3.3, we showed the Bayesian Naïve Bayes model that we use in our Bayesian active learning problem. We use the data obtained from our biochemical application to validate this model. In the application, we were to use the model to predict a peptide is (1) substrate for one type of PPTase, (2) not substrate for the other type of PPTase, and (3) substrate for AcpH. We build three Bayesian Naíve Bayes classifiers, one for each enzyme, to predict whether a peptide is substrate for the given enzyme. After five rounds of experiments, we have tested around 2500 peptides, and obtained binary data of whether the peptides are substrates for the enzymes. We use the binary data to train our classifiers, and use leave-one-out cross validation to gauge the performance of our classifier.

Figure 3.2 shows the performance of our classifiers: albeit the prediction accuracy is not perfect, primarily due to the problem complexity and relatively small amount of data, the model offers moderate amount of prediction power.

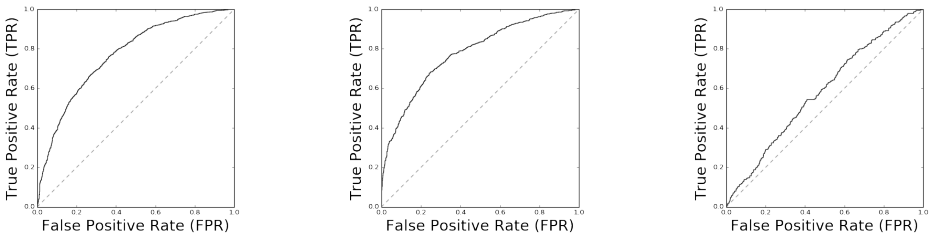


Figure 3.2: ROC curve using leave-one-out cross validation: the left panel is for classification of sfp activity, the middle panel is for classification of AcpS activity, and the right panel is for classification of AcpH activity.

Simulation Study for Comparing Sampling Strategies We also want to compare our sampling strategy to other commonly used strategies in literature, and here we include two of them for benchmark. The first benchmark approach is commonly used in Cheminformatics [81, 2], where the machine learning models

are developed to predict fitness value for molecular substances of interest, and the prediction can in turn be used to find promising substances that maximize the predicted fitness value. Since "prediction" and "optimization" are two separate steps in this approach, we refer to it as the "predict-then-optimize" approach, and note that this method is pure "exploitation". In our biochemical application, the "predict-then-optimize" approach is that we use the Bayesian Naïve Bayes model developed in Section 3.3 to rank the peptides based on their predicted probability to be hits, and pick the top K peptides for testing. The second benchmark method is more of an evolutionary approach [95], where we use the found hits from existing data, and mutate them into new chemical substances. We test the new set, and find out whether the mutation generates new hits. We refer to this method as the "mutation" method, and it is pure "exploration". For our method, we use the procedure proposed in Algorithm 3.1 for the comparison.

To conduct the simulation study, we simulate a system, where for any peptide, whether it is a hit, is completely determined by the Bayesian Naïve Bayes model trained upon all available data, and we refer to it as "the oracle". We then only reveal a small portion of the data to the three methods, and evaluate the quality of the recommended set of peptides by those methods. We measure the quality by computing the probability of finding at least one hit from the recommended set, according to probability computed by "the oracle".

Figure 3.3 shows that our method indeed generates high-quality recommendation sets. It is interesting to see that our method and "predict-then-optimize" method use the same machine learning model, while our method generates much better quality set than "predict-then-optimize" method. To investigate this, we represent the peptides recommended by the three methods in a 2-dimensional

space, where the points are peptides, and the distance is a metric that measures dissimilarity between peptides. Figure 3.4 shows that the peptides recommended by the "predict-then-optimize" method cluster together, indicating that they are very similar to each other. This is expected, since there are so many choices for the peptide design, and when you rank the designs according to their predicted fitness value, the near-top ranking fitness value are expected to be very close to each other, and the underlying peptide designs will also be very similar to each other. In contrast, our method explicitly accounts for the prediction uncertainty in the machine learning model, and instead of optimizing over the predicted quantity, we optimize over the information criterion. The resulting behavior is that, the peptides recommended by our method spread out in the space, exhibiting some exploring behavior. This spreading out pattern also exists in the peptide set recommended by the "mutation" method, however, without a moderate guidance by the machine learning prediction model, the "mutation" method did not achieve great performance by the same exploration effect. Therefore, we have showed that through balancing between "exploitation" and "exploration", our method achieved a better performance over the pure "exploitation" method and the pure "exploration" method.

3.6 Performance in Real Practice

In our biochemical application, we begin with a training set of 14 peptides, extracted from previous reports in literature. Out of the 14 peptides, only a single peptide satisfies both orthogonal labeling and unlabeling criteria, which was found through Phage Display [82], and the other two peptides are orthogonal, but are not substrates for AcpH. Most of the peptides in the initial dataset are longer

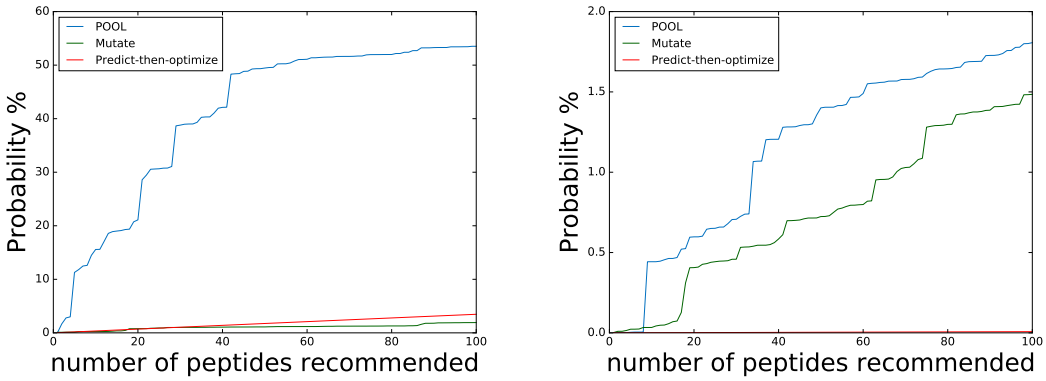


Figure 3.3: Probability of finding at least one hit from the peptide set recommended by the three methods: POOL, mutation, and "predict-then-optimize". The left panel shows the quality of sfp orthogonal peptides, and the right panel shows the quality of AcpS orthogonal peptides.

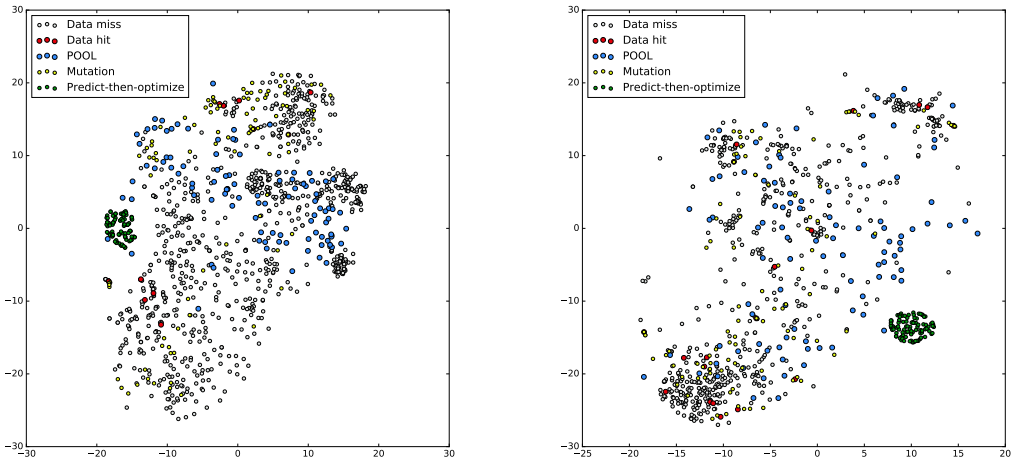


Figure 3.4: 2-dimensional space representation of the peptides recommended by the three methods and the training dataset, where the "hits" in the training data are marked as red. The left panel shows sfp orthogonal peptides, and the right panel shows AcpS orthogonal peptides.

than 40. We applied Algorithm 3.1 to generate a set of 600 peptides as one batch for experimentation, and when the experiments were finished, we obtained binary label of whether the tested peptides were substrates or not, and feed to our machine learning model as new data. We then iteratively did the same procedure for five batches, and tested a total of 2602 peptides. Due to the difficulty of verifying substrate for AcpH, the experimental goal shifted to finding and confirming orthogonal substrates at the end, and we have found 456 orthogonal substrates, of which 333 peptides are orthogonal substrates for Sfp-type PPTase, i.e., they are substrates for Sfp-type PPTase while not for AcpS-type PPTase, and 123 peptides are orthogonal substrates for AcpS-type PPTase. Figure 3.5 shows the histogram of the orthogonal substrates in the distribution of their lengths, and confirms that we have found significant number of short hits.

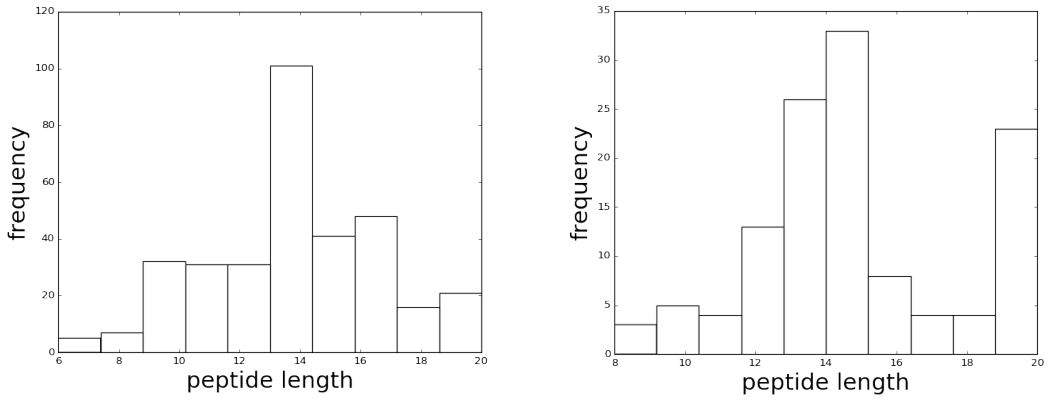


Figure 3.5: Histogram of hits found in real practice: the left panel shows the histogram of Sfp-type orthogonal substrates; the right panel shows the histogram of AcpS-type orthogonal substrates.

3.7 Summary

We proposed an efficient algorithm using greedy heuristic to solve the active learning problem motivated by a biochemical application, and proved that the proposed

algorithm guarantees to achieve at least a factor (1-1/e) to the optimal value. From benchmark results, we further showed that the proposed algorithm outperformed the other two commonly used methods. In addition to the theoretical results and simulation, we demonstrated effectiveness of our method in practice, and helped find an extraordinary number of high-quality hits within the experimental budget, which otherwise would not be possible using traditional approaches.

CHAPTER 4

MULTI-INFORMATION SOURCE OPTIMIZATION AND WARM STARTING BAYESIAN OPTIMIZATION

4.1 Introduction

In this chapter, we consider two closely related problems in Bayesian optimization:

The first is *multi-information source optimization problem* (MISO), where the goal is to optimize a complex design under an expensive-to-evaluate black-box function. To reduce the overall cost we may utilize cheap approximate estimates of the objective that are provided by so-called "information sources". This scenario typically arises in engineering sciences. For example, in aerospace engineering, when designing an airfoil, there are various computer simulators, based on different physical models and with different fidelity, to evaluate the performance of a design. The computer simulators in this context, are information sources, where some of them are more computationally expensive and may provide more accurate estimate of the performance of a airfoil design, and the cheaper information sources may only offer a crude estimate with much less computational cost (see [54]).

The second problem is the warm starting Bayesian optimization, which aims to reduce the solution time required to solve an optimization problem that is one in a sequence of related problems. This problem is important because many businesses that use optimization to make operational decisions solve collections of related optimization problems. For instance, consider a ride-sharing company, such as Uber or Lyft, that makes real-time decisions about how to assign requests for rides to drivers that may fulfill those requests. Such a company might have a single algo-

rithm for making these decisions, whose performance depends crucially on tunable parameters. Given a particular distributional forecast of demand patterns, tuned to the particular next time period, and a simulator to estimate the performance of the algorithm using this forecast, this company might wish to optimize the parameters every few hours to find a configuration that would best achieve their goals in that time interval. Such distributional forecasts would be likely to change across hours within a day and across days within a week due to seasonality, and would also vary from week to week as distributional forecasts include more recent historical data due to changing weather, sporting events, etc.

In both cases, information sources or a sequence of related problems (here we use a single term "auxiliary source" to name them), exhibit certain correlations to the main optimization objective. If we were able to model the correlation between auxiliary sources and the main objective, we have a way to transfer the knowledge learned from those auxiliary sources to the main objective, and therefore save overall cost by efficiently utilizing cheaper information sources, or data obtained from previously solved problems.

Contributions. The main contribution of this work is we built a model that can adaptively learn the degree of correlation between the main objective and the auxiliary sources, and use this information to guide the Bayesian optimization algorithm in the search. In general, we consider the output from the auxiliary sources is not necessarily an unbiased estimator to the main objective, and thus we face a *model discrepancy* which denotes an inherent inability to describe the reality accurately. We stress that this notion goes far beyond typical "noise" such as measurement errors or numerical inaccuracies. The latter grasps uncertainty that arises when sampling from an auxiliary source, and is the type of inaccuracy

that most of the previous work on multifidelity optimization deals with, e.g., see [5, 15, 73, 60, 54] for problem formulations in the engineering sciences. In particular, such an understanding of noise assumes implicitly that the internal value (or state) of the auxiliary source itself is an accurate description of the truth. Our model captures the more general notion of model discrepancy, that accounts for both noise and the uncertainty about the truth that originates from the inherent deviation from reality. We demonstrate the utility of our approach in both settings we mentioned above: the MISO setting, where we optimize the main objective and have access to multiple information sources that are approximate estimates of the true objective; the warm starting setting, which uses prior optimization runs to bootstrap new runs. Our algorithm yields significant improvements in terms of the overall cost of optimization.

Related Work. The work that is closest to our approach is done by Swersky, Snoek, and Adams [87], who showed that the task of tuning hyper-parameters for two classification problems can be sped up significantly by evaluating settings on a small sample instead of the whole database. To this end, they proposed a Gaussian process model to quantify the correlation between such an "auxiliary task" and the primary task, building on previous work on Gaussian process regression for multiple tasks in [7, 29, 88]: their kernel is given by the tensor product $K_t \otimes K_x$, where K_t (K_x) denotes the covariance matrix of the tasks (resp., of the points). They also applied the same model to solving "cold start" problems, treating the previously solved related problems as "auxiliary tasks", and the current problem as the primary task. Their acquisition criterion is a cost-sensitive formulation of Entropy Search [31, 93]: here one samples in each iteration a point that yields a maximum reduction in the uncertainty over the location of the optimum.

Besides, interesting variants of the MISO problem have been studied recently: Kandasamy et al. [44] examined an alternative objective for multifidelity settings that asks to minimize the *cumulative* regret over a series of function evaluations: besides classification problems they also presented an application where the likelihood of three cosmological constants is to be maximized based on Supernovae data. Klein et al. [50] considered hyper-parameter optimization of machine learning algorithms over a large dataset D. Supposing access to subsets of D of arbitrary sizes, they show how to exploit regularity of performance across dataset sizes to significantly speed up the optimization process for support vector machines and neural networks.

In engineering sciences the approach of building cheap-to-evaluate, approximate models for the real function, that offer different fidelity-cost trade-offs, is also known as "surrogate modeling" and has gained a lot of popularity (e.g., see the survey [71]). Kennedy and O'Hagan [47] introduced Gaussian process regression to multifidelity optimization to optimize a design given several computer codes that vary in accuracy and computational complexity. Contrasting the related work discussed above, these articles consider model discrepancy, but impose several restrictions on its nature: a common constraint in multifidelity optimization (e.g., see [47, 5, 15, 73, 60, 30]) is that information sources are required to form a *hierarchy*, thereby limiting possible correlations among their outputs: in particular, once one has queried a high fidelity source for some point x, then no further knowledge on $g(x)$ can be gained by querying any other information source of lower fidelity (at any point). A second frequent assumption is that information sources are *unbiased*, admitting only (typically normally distributed) noise that further must be *independent* across different sources. Lam, Allaire, and Willcox [54] addressed several of these shortcomings by a novel surrogate-based approach that requires

the information sources to be neither hierarchical nor unbiased, and allows a more general notion of model discrepancy building on theoretical foundations by Allaire and Willcox [3]. Their model has a separate Gaussian process for each information source that in particular quantifies its uncertainty over the domain. Predictions are obtained by fusing that information via the method of Winkler [96]. Then they apply the `EI` acquisition function on these surrogates to first decide which design x^* should be evaluated next and afterwards select the respective information source to query x^*; the latter decision is based on a heuristic that aims to balance information gain and query cost.

In the warm starting setting, [16] proposed a meta-learning procedure that improves the performance of a machine learning algorithm by selecting a small set of start-up configurations. The selection procedure explores the new task and then ranks previous tasks (and their corresponding optimal solutions) based on a metric that relies on an evaluation of the "metafeatures" that grasp the characteristics of each dataset. [16] demonstrate that their approach can speed up the process of optimizing hyper-parameters for Support Vector Machines and Algorithm Selection models. This approach differs from ours in three ways: first, the metafeatures and distance functions it develops are specific to machine learning, and extending this approach in a generic way to optimization via simulation problems encountered in operations research would require substantial effort; second, it uses only the final solution from each previous problem, while our approach uses the full set of function evaluations; third, it requires that solutions from previous problems be evaluated under the current objective before they provide useful information, while our approach utilizes this information directly without requiring additional expensive evaluations.

Outline of this Chapter. We first formulate this general problem and discuss our modeling approach in Section 4.2. With our model given, Section 4.3 provides our sampling strategy, which is essentially the cost-sensitive version of the Knowledge Gradient policy; we summarize our algorithm at the end of this section. Section 4.4 demonstrates the performance of our algorithm in empirical analyses; in particular, Section 4.4.1 conducts the experiments under the MISO setting, and Section 4.4.2 shows the result under the warm starting setting.

4.2 Problem Formulation

Each design (or point) x is specified by d parameters. Given some compact set $\mathcal{D} \subset \mathbb{R}^d$ of feasible designs, our goal is to find a best design under some continuous objective function $g : \mathcal{D} \to \mathbb{R}$, i.e. we want to find a design in $\text{argmax}_{x \in \mathcal{D}}\, g(x)$. Restrictions on $\mathcal{D}$ such as box constraints can be easily incorporated in our model. We have access to M possibly biased and/or noisy auxiliary sources $\mathcal{IS}_1, \mathcal{IS}_2, \ldots, \mathcal{IS}_M$ that provide information about g. Note that the $\mathcal{IS}_\ell$ are information sources, or are called "surrogates" in the MISO problems, while in the warm starting problem, they are previously solved related problems. We suppose that observations of $\mathcal{IS}_\ell(x)$ for some ℓ and x are independent and normally distributed with mean value $f(\ell, x)$ and variance $\lambda_\ell(x)$. These sources are thought of as approximating g, with variable model discrepancy or bias $\delta_\ell(x) = g(x) - f(\ell, x)$. We suppose that g can be observed directly without bias (but possibly with noise) and set $\mathcal{IS}_0 = g$; thus, our model is identifiable. Each $\mathcal{IS}_\ell$ is also associated with a query cost function $c_\ell(x) : \mathcal{D} \to \mathbb{R}^+$. We assume that the cost function $c_\ell(x)$ and the variance function $\lambda_\ell(x)$ are both known and continuously differentiable. In practice, these functions may either be provided by domain experts or may be

estimated along with other model parameters from data (see Sect. 4.4, and [74]). Our motivation in having the cost and noise vary over the space of designs is that physical experiments may become difficult to conduct and/or expensive when environmental parameters are extreme. Moreover, simulations may be limited to certain specified parameter settings and their accuracy diminish quickly.

We now place a single Gaussian process prior on f (i.e., on g and the mean response from the M auxiliary sources). Let $\mu : [M] \times \mathcal{D} \mapsto \mathbb{R}$ be the mean function of this Gaussian process, and $\Sigma : ([M] \times \mathcal{D})^2 \mapsto \mathbb{R}$ be the covariance kernel. (Here, for any $a \in \mathbb{Z}^+$ we use $[a]$ as a shorthand for the set $\{1, 2, \ldots, a\}$, and further define $[a]_0 = \{0, 1, 2, \ldots, a\}$.) While our method can be used with an arbitrary mean function and positive semidefinite covariance kernel, we provide a concrete parameterized class of mean function and covariance kernel that are commonly used in Gaussian Process regression, and we will also detail how to estimate the hyper-parameters of the mean function and the covariance kernel in the later paragraph.

Independent Model Discrepancy. We first propose a parameterized class of mean functions μ and covariance functions Σ derived by supposing that model discrepancies are chosen independently across auxiliary sources. This first approach is appropriate when auxiliary sources are different in kind from each other and share no relationship except the fact that they are modeling a common objective.

We suppose here that δ_ℓ for each $\ell > 0$ was drawn from a separate independent Gaussian process, $\delta_\ell \sim GP(\mu_\ell, \Sigma_\ell)$. We also suppose that δ_0 is identically 0, and that $f(0, \cdot) \sim GP(\mu_0, \Sigma_0)$, for some given μ_0 and Σ_0. We then define $f(\ell, x) = f(0, x) + \delta_\ell(x)$ for each $\ell \in [M]$. Typically, one would not have a strong prior

belief as to the direction of the bias inherent in an auxiliary source, and so we set $\mu_\ell(x) = 0$. (If one does have a strong prior opinion that an auxiliary source is biased in one direction, then one may instead set μ_ℓ to a constant estimated using maximum a posteriori estimation.) With this modeling choice, we see that the mean of $f \sim GP(\mu, \Sigma)$ with mean function μ and covariance kernel Σ is given by $\mu(\ell, x) = \mathbb{E}[f(\ell, x)] = \mathbb{E}[f(0, x)] + \mathbb{E}[\delta_\ell(x)] = \mu_0(x)$ for each $\ell \in [M]_0$, since $\mathbb{E}[\delta_\ell(x)] = 0$ holds. Additionally, for $\ell, m \in [M]_0$ and $x, x' \in \mathcal{D}$,

$$\begin{aligned}
&\Sigma((\ell, x), (m, x')) \\
&= \mathrm{Cov}(f(0, x) + \delta_\ell(x), f(0, x') + \delta_m(x')) \\
&= \Sigma_0(x, x') + \mathbb{1}_{\ell,m} \cdot \Sigma_\ell(x, x'),
\end{aligned}$$

where $\mathbb{1}_{\ell,m}$ denotes Kronecker's delta, and where we have used independence of δ_ℓ, δ_m, and $f(0, \cdot)$.

Correlated Model Discrepancies We also propose a more general parameterized class that models correlation between model discrepancies, as is typical when auxiliary sources can be partitioned into groups, such that the sources within a group tend to agree more amongst themselves than they do with the sources in other groups. This case arises when a sequence of optimization problems were solved over time, and exhibit correlation over time. In engineering sciences we witness this if some sources share a common modeling approach, as for example, if one set of sources for an airfoil modeling problem correspond to different discretizations of a PDE that models wing flutter, while another set provides various discretizations of another PDE that modeling airflow. Two sources that solve the same PDE will be more correlated than two that solve different PDEs.

Formally, let $P = \{P_1, \ldots, P_Q\}$ denote a partition of $[M]$ and define the func-

tion $k : [M] \to [Q]$ that gives for each IS its corresponding partition in P. Let there be an independent Gaussian process $\varepsilon(k(\ell), x) \sim GP(\mu_{k(\ell)}, \Sigma_{k(\ell)})$ for each partition. Again our approach is to incorporate all Gaussian processes into a single one with prior distribution $f \sim GP(\mu, \Sigma)$:[1] therefore, for all $\ell \in [M]_0$ and $x \in \mathcal{D}$ we define $f(\ell, x) = f(0, x) + \varepsilon(k(\ell), x) + \delta_\ell(x)$, where $f(0, x) = g(x)$ is the objective function that we want to optimize. Due to linearity of expectation, we have

$$\begin{aligned}\mu(\ell, x) &= \mathbb{E}\left[f(0, x) + \varepsilon(k(\ell), x) + \delta_\ell(x)\right] \\ &= \mathbb{E}\left[f(0, x)\right] + \mathbb{E}\left[\varepsilon(k(\ell), x)\right] + \mathbb{E}\left[\delta_\ell(x)\right] \\ &= \mu_0(x),\end{aligned}$$

since $\mathbb{E}\left[\varepsilon(k(\ell), x)\right] = \mathbb{E}\left[\delta_\ell(x)\right] = 0$. Recall that the indicator variable $\mathbb{1}_{\ell,m}$ denotes Kronecker's delta. Let $\ell, m \in [M]_0$ and $x, x' \in \mathcal{D}$, then we define the following composite covariance function Σ:

$$\begin{aligned}&\Sigma\left((\ell, x), (m, x')\right) \\ &= \mathrm{Cov}\left(f(0, x) + \varepsilon(k(\ell), x) + \delta_\ell(x), f(0, x')\right. \\ &\quad \left. + \varepsilon(k(m), x') + \delta_m(x')\right) \\ &= \mathrm{Cov}(f(0, x), f(0, x')) + \mathrm{Cov}(\varepsilon(k(\ell), x), \varepsilon(k(m), x')) \\ &\quad + \ \mathrm{Cov}(\delta_\ell(x), \delta_m(x')) \\ &= \Sigma_0(x, x') + \mathbb{1}_{k(\ell),k(m)} \cdot \Sigma_{k(\ell)}(x, x') + \mathbb{1}_{\ell,m} \cdot \Sigma_\ell(x, x').\end{aligned}$$

Estimation of Hyper-Parameters In the case of independent model discrepancy, one would suppose that the functions $\mu_0(\cdot)$ and $\Sigma_\ell(\cdot, \cdot)$ with $\ell \in [M]_0$ belong to some parameterized class: for example, one might set $\mu_0(\cdot)$ and each $\lambda_\ell(\cdot)$ to constants, and suppose that Σ_ℓ each belong to the class of Matérn kernels [74,

[1] For simplicity we reuse the notation from the first model to denote their pendants in this model.

Section 4.2].

The hyper-parameters of the covariance kernel are fit from data. In typical Bayesian optimization problems, it is common that we face the sparsity of training data, and from the author's experience, the maximum likelihood estimates (MLE) is often less robust than *maximum a posteriori* (MAP) estimation, due to the MAP's additional ability of encoding prior belief about the distribution of the hyper-parameters. Therefore, we choose to use MAP to estimate the hyper-parameters from the data.

For a Matérn kernel we have to estimate $d + 1$ hyper-parameters for each auxiliary source: d length scales, and the signal variance. We suppose a normal prior $\mathcal{N}(\mu_i, \sigma_i^2)$ for hyper-parameter θ_i. Let $D \in \mathcal{D}$ be a set of points, for example chosen via a Latin Hypercube design, and evaluate every auxiliary source at all points in D. We estimate the hyper-parameters for $f(0, \cdot)$ and the δ_i for $i \in [M]$, using the "observations" $\Delta_i = \{\mathcal{IS}_i(x) - \mathcal{IS}_0(x) \quad | \quad x \in D\}$ for the δ_i. The prior mean of the signal variance parameter of $\mathcal{IS}_0$ is set to the variance of the observations at $\mathcal{IS}_0$ minus their average observational noise. The mean for the signal variance of $\mathcal{IS}_i$ with $i \in [M]$ is obtained analogously using the "observations" in Δ_i; here we subtract the mean noise variance of the observations at $\mathcal{IS}_i$ and the mean noise at $\mathcal{IS}_0$, exploiting the assumption that observational noise is independent. Regarding the means of the priors for length scales, we found it useful to set each prior mean to the length of the interval that the corresponding parameter is optimized over. For all hyper-parameters θ_i we set the variance of the prior to $\sigma_i^2 = (\frac{\mu_i}{2})^2$, where μ_i is the mean of the prior.

For the case of correlated model discrepancies, we are able to follow the very similar guideline using simple analogy.

How to Express Beliefs on Fidelities of Auxiliary Sources In many applications one has beliefs about the relative accuracies of the auxiliary sources. One approach to explicitly encode these is to introduce a new coefficient α_ℓ for each Σ_ℓ that typically would be fitted from data along with the other hyper-parameters. But we may also set it at the discretion of a domain expert, which is particularly useful if none of the sources is an unbiased estimator and we rely on regression to estimate the true objective. For squared exponential kernel, this coefficient is sometimes part of the formulation and referred to as "signal variance" (e.g., see [74, p. 19]). For the sake of completeness, we detail the effect for our simple model with independent model discrepancies. Recall that we suppose $f \sim GP(\mu, \Sigma)$ with a mean function μ and covariance kernel Σ, and observe that the introduction of the new coefficient α_ℓ does not affect $\mu(\ell, x)$. However, it changes $\Sigma((\ell, x), (m, x'))$ to

$$\Sigma((\ell, x), (m, x')) = \Sigma_0(x, x') + \mathbb{1}_{\ell,m} \cdot \alpha_\ell \cdot \Sigma_\ell(x, x').$$

We observe that setting α_ℓ to a larger value results in a bigger uncertainty. The gist is that then samples from such source have less influence in the Gaussian process regression (e.g., see Eq. (A.6) on pp. 200 in [74]). It is instructive to consider the case that we observe a design x at a noise-free source: its observed output coincides with $f(\ell, x)$ (with zero variance). Our estimate $f(0, x)$ for $g(x)$, however, is a Gaussian random variable whose variance depends (in particular) on the uncertainty of the above information source as encoded in α_ℓ, since $\lambda_\ell(x) = 0$ holds.

4.3 The Sampling Strategy

Our optimization algorithm proceeds in rounds, where in each round it selects a design $x \in \mathcal{D}$ and a source $\mathcal{IS}_\ell$ with $\ell \in \mathcal{L}$, where $\mathcal{L}$ is the set of all accessible sources. For example, in the MISO problems, we may set $\mathcal{L} = [M]_0$, while in the warm starting problems, $\mathcal{L} = [0]$, because all the previously solved problems are useful in providing additional information from their data, but are probably not accessible for sampling new points. The goal is to find an x that maximizes $g(x)$ over $\mathcal{D}$. Let us assume for the moment that the query cost is uniform over the whole domain and all sources; we will show how to remove this assumption later. Further, assume that we have already sampled n points X and obtained the observations Y. Finally, denote by $\mathbb{E}_n[f(\ell, x)]$ the expected value according to the posterior distribution given X and Y and shorthand $\mu^{(n)}(\ell, x) := \mathbb{E}_n[f(\ell, x)]$. Since that distribution is normal, the best *expected* objective value of any design, as estimated by our statistical model, is $\max_{x' \in \mathcal{D}} \mathbb{E}_n[f(0, x')] = \max_{x' \in \mathcal{D}} \mu^{(n)}(0, x')$. If we were to pick an $x \in \mathcal{D}$ now irrevocably, then we would select an x of maximum expectation, i.e. $x \in \operatorname{argmax}_{x' \in \mathcal{D}} \mu^{(n)}(0, x')$. This motivates choosing the next design $x^{(n+1)}$ and information source $\ell^{(n+1)}$ that we will sample such that we maximize $\mathbb{E}_n\left[\max_{x' \in \mathcal{D}} \mu^{(n+1)}(0, x') \;\middle|\; \ell^{(n+1)} = \ell, x^{(n+1)} = x\right]$, or equivalently maximize the expected *gain* over the current optimum by $\mathbb{E}_n\left[\max_{x' \in \mathcal{D}} \mu^{(n+1)}(0, x') \;\middle|\; \ell^{(n+1)} = \ell, x^{(n+1)} = x\right] - \max_{x' \in \mathcal{D}} \mu^{(n)}(0, x')$. Note that the equivalence of the maximizers for both expressions follows immediately from the observation that $\mathbb{E}_n\left[\mu^{(n)}(0, x')\right] = \mu^{(n)}(0, x')$ is a constant for all $x' \in \mathcal{D}$ given X and Y.

Next we show how the assumption made at the beginning of this section, that query cost are uniform across the domain and for all sources, can be removed. To

this end, we associate a query cost function $c_\ell(x) : \mathcal{D} \to \mathbb{R}^+$ with each source $\mathcal{IS}_\ell$ for $\ell \in \mathcal{L}$. Then our goal becomes to find a sample $(\ell^{(n+1)}, x^{(n+1)})$ whose value of information divided by its respective query cost is maximum. The gist is that conditioned on any $(\ell^{(n+1)}, x^{(n+1)})$, the expected gain of all $x' \in \mathcal{D}$ is scaled by $c_{\ell^{(n+1)}}(x^{(n+1)})^{-1}$. Then the *cost-sensitive Knowledge Gradient* policy picks a sample (ℓ, x) that maximizes the expectation

$$\mathbb{E}_n \left[\frac{\max_{x' \in \mathcal{D}} \mu^{(n+1)}(0, x') - \max_{x' \in \mathcal{D}} \mu^{(n)}(0, x')}{c_\ell(x)} \;\middle|\; \ell^{(n+1)} = \ell, x^{(n+1)} = x \right], \quad (4.1)$$

denoted by $\mathrm{CKG}(\ell, x)$. Our task is to compute $(\ell^{(n+1)}, x^{(n+1)}) \in \operatorname{argmax}_{\ell \in \mathcal{L}, x \in \mathcal{D}} \mathrm{CKG}(\ell, x)$, which is a nested optimization problem.

To make this task feasible in practice, we discretize the domain of the inner maximization problem stated in Eq. (4.1): for simplicity, we choose the discrete set $\mathcal{A} \subset \mathcal{D}$ via a Latin Hypercube design. Now we have reduced the inner optimization problem for each information source to the setting of Frazier et al. [20] who showed how to compute the value of information over a discrete set if there is only one information source and query costs are uniform.

We provide an outline and refer to their article for details. For their setting let $\bar{\mu}^n$ be the vector of posterior means for $\mathcal{A}$ and define for each $x \in \mathcal{A}$ $\bar{\sigma}^n(x) = \Sigma^n e_x / (\lambda(x) + \Sigma^n_{xx})$, where Σ^n is the covariance matrix of the posterior distribution and e_x is the vector that has a one at the entry corresponding to x and zeros elsewhere. Given these vectors, observe that

$$\begin{aligned} &\mathbb{E}_n \left[\max_{x' \in \mathcal{A}} \mu^{(n+1)}(0, x') - \max_{x' \in \mathcal{A}} \mu^{(n)}(0, x') \;\middle|\; x^{(n+1)} = x \right] \\ &= h\left(\bar{\mu}^n, \bar{\sigma}^n(x)\right), \end{aligned}$$

where $h(a, b) = \mathbb{E}_n \left[\max_i a_i + b_i Z \right] - \max_i a_i$ for vectors a, b, and Z is a one-dimensional standard normal random variable. Frazier et al. show how to com-

pute h efficiently.

Thus, following our initial considerations, we approximate the cost-sensitive Knowledge Gradient by maximizing $\frac{h(\bar{\mu}^n,\bar{\sigma}^n(\ell,x))}{c_\ell(x)}$ over $\mathcal{L}\times\mathcal{D}$, i.e. the outer optimization problem is still formulated over $\mathcal{D}$. Note that we can compute the gradient of $\frac{h(\bar{\mu}^n,\bar{\sigma}^n(\ell,x))}{c_\ell(x)}$ with respect to x assuming that c_ℓ is differentiable (e.g., when given by a suitable Gaussian process). Thus, we may apply a multi-start gradient-based optimizer to compute $(\ell^{(n+1)}, x^{(n+1)})$.

We summarize our algorithm:

1. Using samples from all sources, estimate hyper-parameters of the Gaussian process prior as described Section 4.2.

   Then calculate the posterior f based on the prior and samples.

2. Until the budget for samples is exhausted do:

   Determine the source $\ell\in\mathcal{L}$ and the design $x\in\mathcal{D}$ that maximize the cost-sensitive Knowledge Gradient proposed in Eq. (4.1) and observe $\mathcal{IS}_\ell(x)$.

   Update the posterior distribution with the new observation.

3. Return the point with the largest estimated value according to the current posterior $f(0,\cdot)$.

4.4 Numerical Experiments

We conducted numerical experiments under both MISO and warm starting settings, to demonstrate the performance of our algorithm.

4.4.1 The MISO Problems

We compare our algorithm under the MISO setting, called `misoKG`, to the state-of-the-art Bayesian optimization algorithms for MISO problems. The statistical model and the sampling strategy were implemented in `Python 2.7` and `C++` using the functionality provided by `Metrics Optimization Engine` [12], in particular for Gaussian process regression and fitting hyper-parameters.

Benchmark Algorithms. The first benchmark method, `MTBO+`, is an improved version of Multi-Task Bayesian Optimization proposed by Swersky et al. [87]. It uses a cost-sensitive version of Entropy Search as acquisition function that picks samples to maximize the information gain over the location of the optimum of the objective function, normalized by the respective query cost (see their paper for details). `MTBO` combines acquisition function with a "multi-task" Gaussian process model that captures the relationships between information sources (called "tasks") and the objective function (see their paper for details). Following a recommendation of Snoek [84], our implementation `MTBO+` uses an improved formulation of the acquisition function given by [32, 85], but otherwise is identical to `MTBO`; in particular, it uses the statistical model of [87].

The other algorithm, `misoEI` of [54], was developed to solve MISO problems that involve model discrepancy and therefore is a good competing method to compare with. It maintains a separate Gaussian process for each information source: to combine this knowledge, the corresponding posterior distributions are fused for each design via Winkler's method [96] into a single intermediate surrogate, which is a normally distributed random variable. Then Lam et al. adapt the Expected Improvement (`EI`) acquisition function to select the design which is to be sam-

pled next: for the sake of simplicity, assume that observations are noiseless and that y^* is the objective value of a best sampled design. If Y_x denotes a Gaussian random variable with the posterior distribution of the objective value for design x, then $\mathbb{E}[\max\{Y_x - y^*, 0\}]$ is the expected improvement for x, and the `EI` acquisition function selects an x that maximizes this expectation. Based on this decision, the information source to invoke is chosen by a heuristic that aims at maximizing the `EI` per unit cost.

The Experimental Setups. We conducted numerical experiments on the following test problems: the first is the 2-dimensional Rosenbrock function which is a standard benchmark in the literature, tweaked into the MISO setting by [54]. The second is a MISO benchmark proposed by [87]: the goal is to optimize the four hyper-parameters of a machine learning algorithm, using a small, related set of smaller images as cheap information source. The third is an assemble-to-order problem introduced by Hong and Nelson [37]: here the objective is to optimize an 8-dimensional target stock vector in order to maximize the expected daily profit of a company, for which an estimate is provided as an output by their simulator.

In MISO settings the amount of initial data that one can use to inform the methods about each information source is typically dictated by the application, in particular by resource constraints and the availability of the respective source. In our experiments all methods were given *identical initial datasets* for each information source in every replication; these sets were drawn randomly via Latin Hypercube designs. For the sake of simplicity, we provided the same number of points for each $\mathcal{IS}$, deliberately set in advance to 2.5 point per dimension of the design space $\mathcal{D}$. In particular, this choice is *not geared* towards particular benchmark methods. Regarding the kernel and mean function, `MTBO+` uses the settings

provided in [85]. The other algorithms used the squared exponential kernel and a constant mean function set to the mean of a sample.

We report the "gain" over the best initial solution, that is the true objective value of the respective design that a method would return at each iteration minus the best value in the initial data set. If the true objective value is not known for a given design, we report the value obtained from the information source of highest fidelity. This gain is plotted as a function of the *total cost*, that is the cumulative cost for invoking the information sources plus the fixed cost for the initial data; this metric naturally generalizes the number of function evaluations prevalent in Bayesian optimization. Note that the computational overhead of choosing the next information source and sample is omitted, as it is negligible compared to invoking an information source in real-world applications. Error bars are shown at the mean plus and minus *two standard errors* averaged over at least 100 runs of each algorithm. Note that even for deterministic sources a tiny observational noise of 10^{-6} is supposed to avoid numerical issues during matrix inversion.

The Rosenbrock Benchmarks We consider the design space $\mathcal{D} = [-2, 2]^2$, and $M = 2$ information sources. $\mathcal{IS}_0$ is the Rosenbrock function $f(\cdot)$ plus optional Gaussian noise, and $\mathcal{IS}_1$ equals $f(\cdot)$ with an additional oscillatory component:

$$\begin{aligned} f(\boldsymbol{x}) &= (1 - x_1)^2 + 100 \cdot (x_2 - x_1^2)^2 \\ \mathcal{IS}_0(\boldsymbol{x}) &= f(\boldsymbol{x}) + u \cdot \varepsilon \\ \mathcal{IS}_1(\boldsymbol{x}) &= f(\boldsymbol{x}) + v \cdot \sin(10 \cdot x_1 + 5 \cdot x_2) \end{aligned} \tag{4.2}$$

where $\boldsymbol{x} = (x_1, x_2)^T \in \mathcal{D}$, and ε is i.i.d. noise drawn from the standard normal distribution. Moreover, u and v are configuration constants that can vary when running experiments in different settings. We suppose that $\mathcal{IS}_1$ is not subject to

observational noise, hence the uncertainty only originates from the model discrepancy. We experimented on two different configurations to gain a better insight into characteristics of the algorithms. Since Lam et al. [54] reported a good performance of their method on (4.2), we replicated their experiment using the same parameters to compare the performance of the four methods: that is, we set $u = 0$, $v = 0.1$. Replicating the setting in [54, p. 15], we also suppose a tiny uncertainty for $\mathcal{IS}_0$, although it actually outputs the truth, and set $\lambda_0(x) = 10^{-3}$ and $\lambda_1(x) = 10^{-6}$ for all x. Furthermore, we assume a cost of 1000 for each query to $\mathcal{IS}_0$ and of 1 for $\mathcal{IS}_1$.

Since all methods converged to good solutions quickly, we investigate the ratio of gain to cost: Fig. 4.4 (t) displays the gain of each method over the best initial solution as a function of the total cost inflicted by querying information sources. We see that the new method `misoKG` offers a significantly better gain per unit cost. A closer analysis shows that `misoKG` finds an almost optimal solution typically within $5 - 10$ samples. Interestingly, `misoKG` relies only on cheap samples, therefore managing the uncertainties successfully. `MTBO+` on the other hand struggles initially, but then eventually obtains a near-optimal solution, too. To this end, it makes usually one or two queries of the expensive truth source after about 40 steps. `misoEI` shows a odd behavior: it takes several queries, one of them to $\mathcal{IS}_0$, before it gains over the best initial design for the first time. Then it jumps to a very good solution and subsequently samples only the cheap $\mathcal{IS}$.

For the second setup, we make the following changes: we set $u = 1$ and $v = 2$, and suppose for $\mathcal{IS}_0$ uniform observational noise of $\lambda_0(x) = 1$ and uniform query cost $c_0(x) = 50$. Note that now the difference of the costs of both sources is much smaller, while their uncertainties (and hence the variances) are considerably

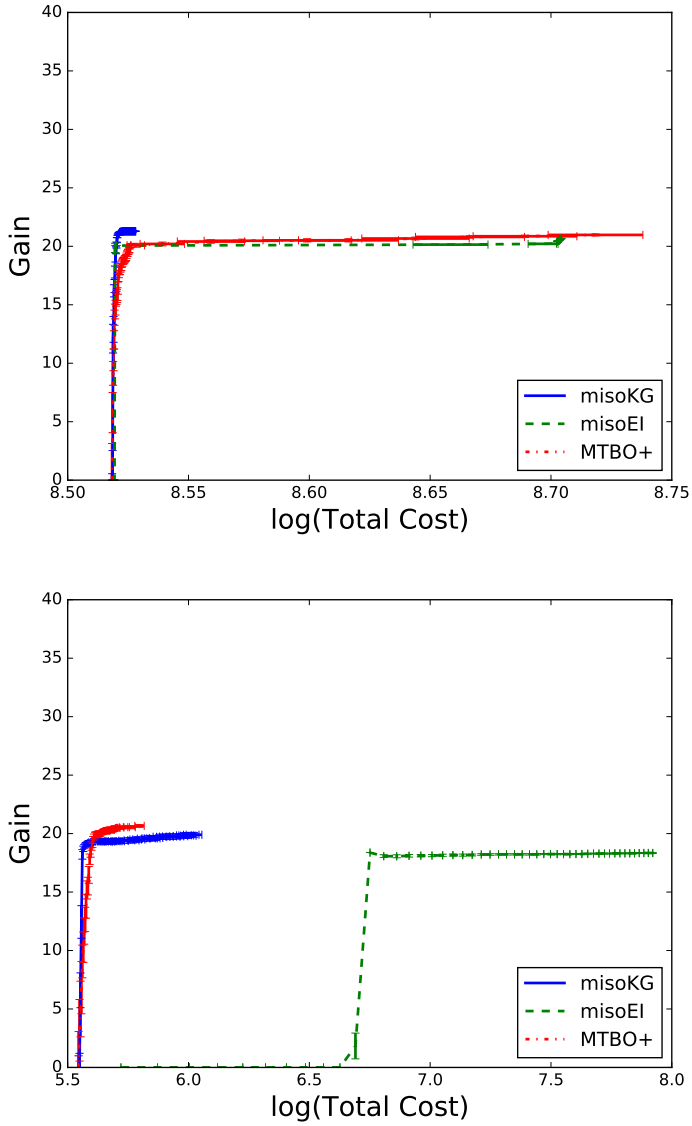


Figure 4.1: (t) The Rosenbrock benchmark with the parameter setting of [54]: `misoKG` offers an excellent gain-to-cost ratio and outperforms its competitors substantially. (b) The Rosenbrock benchmark with the alternative setup.

bigger. The results are displayed in Fig. 4.1 (b): as for the first configuration, `misoKG` outperforms the other methods. Interestingly, `misoEI`'s performance is drastically decreased compared to the first choice of parameters, since it only queries the expensive truth. Looking closer, we see that `misoKG` initially queries only the cheap information source $\mathcal{IS}_1$ until it comes close to an optimal value after about five samples. It starts to query $\mathcal{IS}_0$ occasionally later. As on the first Rosenbrock benchmark, `MTBO+` starts slower than `misoKG`. But this time it catches up and indeed surpasses `misoKG` slightly. `MTBO+` invokes $\mathcal{IS}_0$ once after about $40 - 45$ steps.

The Image Classification Benchmark This classification problem was introduced by Swersky, Snoek, and Adams [87] to demonstrate the performance of their `MTBO` algorithm. The goal is to optimize four hyper-parameters of the Logistic Regression algorithm [89] (the learning rate, the penalty coefficient of the L2-regularization, the batch size, and the number of epochs) in order to minimize the classification error on the MNIST dataset of [55]. The MNIST dataset contains 70,000 grayscale images of handwritten digits, where each image consists of 784 pixels. In the experimental setup information source $\mathcal{IS}_0$ corresponds to invoking the machine learning algorithm on this dataset. Following Swersky et al., the USPS dataset serves as cheap information source $\mathcal{IS}_1$: this set comprises only about 9000 images of handwritten digits that are also smaller, only 256 pixels each [91]. Again we suppose a tiny observational noise of 10^{-6} and set the invocation cost of the sources to 4.5 for $\mathcal{IS}_1$ and 43.69 for $\mathcal{IS}_0$ respectively. Both information sources report the logit of the test error achieved on their respective data set. A closer examination shows that $\mathcal{IS}_1$ is subject to considerable bias with respect to $\mathcal{IS}_0$, making it a challenge for MISO algorithms.

Initially, misoKG and MTBO+ are on par and both outperform misoEI (cp. Fig.4.2 (t)). In order to study the convergence behavior, we evaluated misoKG and MTBO+ for 150 steps, with a lower number of replications but using the same initial data for each replication. We observe that misoKG converges to the global optimum, usually after about 80 queries to information sources (see Fig.4.2 (b)). In its late iterations MTBO+ achieves a lower performance than misoKG has at this respective cost. Note that the experimental results of [87] show that MTBO+ will also converge to the optimum eventually.

It is quite interesting to see how the algorithms respond to the model discrepancy of $\mathcal{IS}_1$: misoKG and MTBO+ initially only query $\mathcal{IS}_1$, but then soon mix in queries to $\mathcal{IS}_0$ periodically. misoKG makes more queries to $\mathcal{IS}_0$ than MTBO+. misoEI, however, directs the overwhelming majority of its queries to $\mathcal{IS}_1$ and seems to experience difficulties in capitalizing from these observations.

The Assemble-To-Order Benchmark In the assemble-to-order (ATO) benchmark, a reinforcement learning problem from a business application, we are managing the inventory of a company that manufactures m products. Each of these products is made from a selection from n items, where we distinguish for each product between key items and non-key items: if the company runs out of key items, then it cannot sell the respective products until it has restocked its inventory; non-key items are optional and used if available. When the company sends a replenishment order, the required item is delivered after a random period whose distribution is known. Since items in the inventory inflict holding cost, our goal is to find an optimal target inventory level vector b that determines the amount of each item we want to stock, such that we maximize the expected profit per day (see [37] for details).

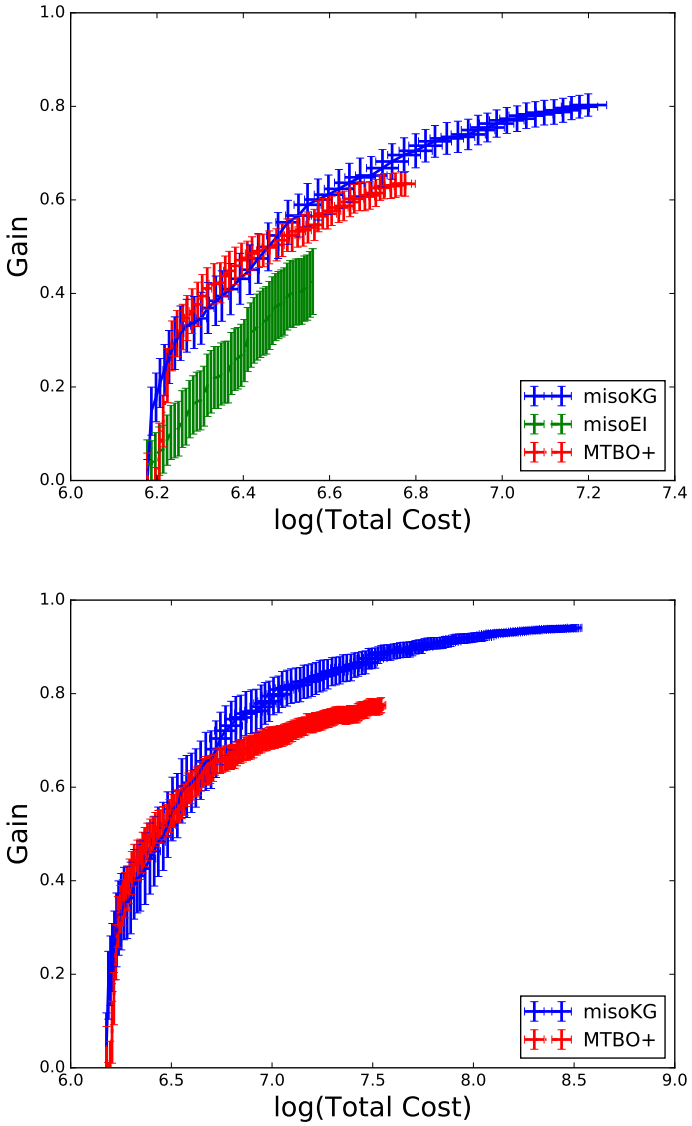


Figure 4.2: The performance on the image classification benchmark with significant model discrepancy. (t) The first 50 steps of each algorithm: `misoKG` and `MTBO+` perform better than `misoEI`. (b) The first 150 steps of `misoKG` and `MTBO+`. While the initial performance of `misoKG` and `MTBO+` is comparable, `misoKG` achieves better testscores after about 80 steps and converges to the global optimum.

Hong and Nelson proposed a specific scenario with m=5 different products that depend on a subset of n=8 items, thus our task is to optimize the 8-dimensional target vector $b \in [0, 20]^8$. For each such vector their simulator provides an estimate of the expected daily profit by running the simulation for a variable number of replications (see #runs in Table 4.1). Increasing this number yields a more accurate estimate but also has higher computational cost. The observational noise and query cost, i.e. the computation time of the simulation, are estimated from samples for each information source; for the sake of simplicity assuming that both functions are constant over the domain.

There are two simulators for this assemble-to-order setting that differ subtly in the model of the inventory system. However, the effect in estimated objective value is significant: on average the outputs of both simulators at the same target vector differ by about 5% of the score of the global optimum, which is about 120, whereas the largest observed bias out of 1000 random samples was 31.8. Moreover, the sampled variance of the difference between the outputs of both simulators is about 200. Thus, we are witnessing a significant model discrepancy. We set up three information sources (cp. Table 4.1): $\mathcal{IS}_0$ and $\mathcal{IS}_2$ use the simulator of Xie et al. [98], whereas the cheapest source $\mathcal{IS}_1$ invokes the implementation of Hong and Nelson. We assume that $\mathcal{IS}_0$ models the truth. The performance of the algorithms

Table 4.1: The Parameters for the ATO problem

	# Runs	Noise Variance	Cost
$\mathcal{IS}_0$	500	0.056	17.1
$\mathcal{IS}_1$	10	2.944	0.5
$\mathcal{IS}_2$	100	0.332	3.9

is summarized in Fig. 4.3. `misoKG` achieves the best gain-to-cost ratio of all three algorithms. Interestingly, `misoKG` and `MTBO+` utilize in their optimization of the

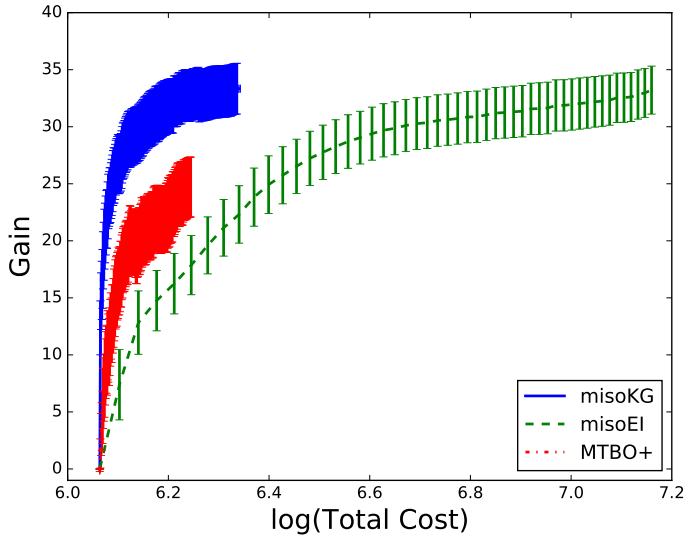


Figure 4.3: The performance on the assemble-to-order benchmark with significant model discrepancy. `misoKG` has the best gain per cost ratio among the algorithms.

target inventory level vector only the cheapest, heavily biased source, and therefore are able to obtain significantly better gain per cost ratios than their competitors. Fig. 4.3 displays the performance over the first 150 steps for `misoKG` and `MTBO+` and the first 50 steps of `misoEI`, all averaged over 100 runs. `misoKG` has the best start and dominates in particular `MTBO+` clearly. `misoKG` averages at a gain of 26.1, but inflicts only an average query cost of 54.6 to the information sources, excluding the fixed cost for the initial datasets that are identical for all algorithms. This is only 6.3% of the query cost that `misoEI` requires to achieve a comparable score. `misoEI` that employs a two-step heuristic for trading off predicted gain and query cost chose almost always the most expensive simulation to evaluate the selected design.

4.4.2 The Warm Starting Problems

To demonstrate the speedup we could achieve by utilizing previous runs, we use the warm starting version of our algorithm `wsKG`, and compare with two other baseline methods, that are both Bayesian optimization algorithms, except that they do not utilize previous runs.

Benchmark Algorithms. The first baseline method is the well-known `EGO` algorithm of [42]. `EGO` is also a myopic algorithm that iteratively selects one point to sample. `EGO`'s acquisition criterion is to optimize the expected improvement (`EI`), which has been reviewed in Section 1.2.2.

In order to assess the benefit of taking data on previous runs into account, we also compare `wsKG` to the "vanilla version" of the Knowledge Gradient (`KG`); see Section 1.2.2 for a review on KG. In this comparison, since both algorithms use the same kind of sampling policy, i.e., `KG`, any speedup is purely benefited from the additional data from previous runs.

The Experimental Setups. We conducted experiments on two suites of test problems: the first comprises variants of the two-dimensional Rosenbrock function; the second family of testbed instances are taken from the Assemble-To-Order (`ATO`) problem. We have introduced both test problems in the benchmarks for the MISO setting.

In this evaluation all algorithms are given the same initial data points drawn randomly for each instance and each run. `wsKG` is additionally provided with one data set for the Rosenbrock instances and two for `ATO`: each set contains the samples

collected during a single run on a related instance. A single run consists of 25 steps (also referred to as iterations) for each of the Rosenbrock instances and 50 steps for the Assemble-to-Order suite.

The hyper-parameters of the Gaussian Process affect how accurate the posterior distribution predicts the objective function of the current task. In accordance with our scenario, we optimized the hyper-parameters once for a single instance of each suite and then invoked `wsKG` with this fixed setting for all instances. For the baseline methods `EGO` and `KG` we optimized these parameters for each instance in advance, thereby possibly giving them an edge over `wsKG` in this respect.

The plots below provide the mean of the gain over the initial solution for each iteration. Error bars are shown at the mean plus and minus two standard errors, averaged over at least 100 replications.

The Rosenbrock Family The basis of these instances is the 2D Rosenbrock function $RB_1(x_1, x_2) = (1 - x_1)^2 + 100 \cdot (x_2 - x_1^2)^2$, which subject to the following modifications:

$$RB_2(x_1, x_2) = RB_1(x_1, x_2) + .01 \cdot \sin(10 \cdot x_1 + 5 \cdot x_2)$$
$$RB_3(x_1, x_2) = RB_1(x_1 + .01, x_2 - .005)$$
$$RB_4(x_1, x2) = RB_2(x_1, x_2) + .01 \cdot x_1$$

Moreover, each function evaluation is subject to i.i.d. noise with mean zero and variance .25. The task is to optimize the respective function on the domain $[-2, 2]^2$.

The results are summarized in Fig. 4.4: `wsKG` obtains a near-optimal solution already after only one or two samples, thanks to its additional information from the previous run. `KG` and `EGO` converge slower, with `KG` showing a superior performance

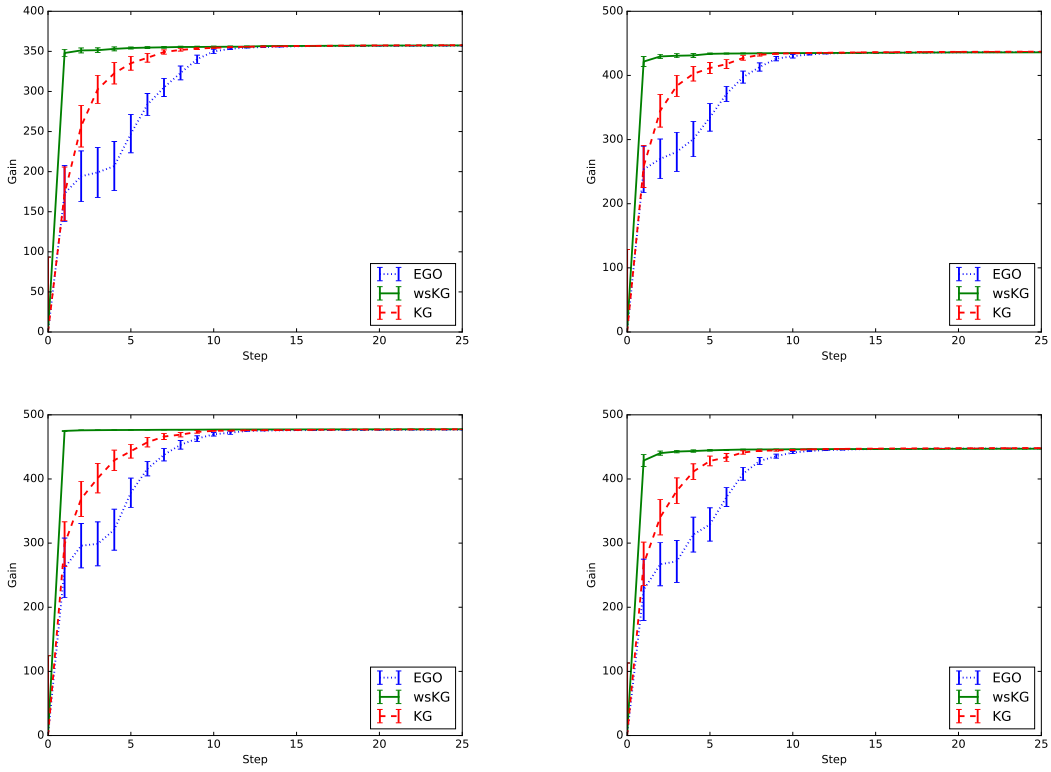


Figure 4.4: (ul) The basic Rosenbrock function RB_1. (ur) The Rosenbrock function RB_2 with an additive sine. (bl) The shifted Rosenbrock function RB_3. (br) The Rosenbrock function RB_4 with an additive sine and a bias depending on x_1.

over `EGO`.

The Assemble-To-Order Benchmark In the original Assemble-To-Order problem, [37] proposed a setting (referred to as `ATO` 1) for $n = 8$ items, $m = 5$ products, and search domain $[0, 20]^8$. To simulate the warm starting setting, we have created three additional instances based on `ATO` 1:

In the first time period `ATO` 2 the forecast sees a change in the customer behavior: the expected demand for the two products that had been most popular before drops by $3 - 5\%$. On the other hand, the popularity of two other products grows by up to 5%. Additionally, the profit of some of the items increases by $1 - 2\%$ on

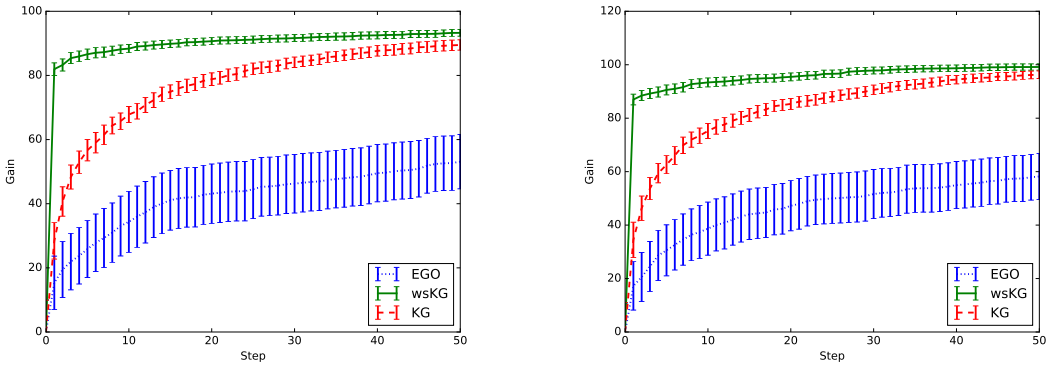


Figure 4.5: (l) `ATO` 1 (r) `ATO` 2: All algorithms have the same initial data for the current problem. `wsKG` has also access to samples of two runs on related instances, but its hyper-parameters are not optimized for the current instance.

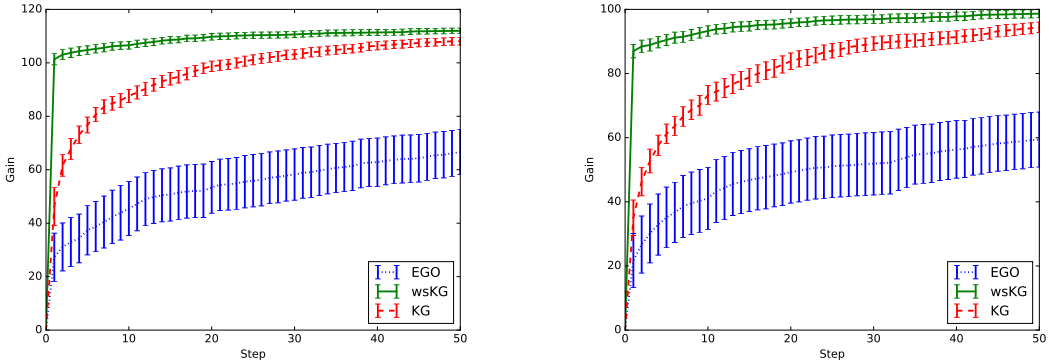


Figure 4.6: (l) `ATO` 3 (r) `ATO` 4: `wsKG` has received the samples of two runs on related instances.

average.

In the next period `ATO` 3 the delivery of some of the items is delayed; the expected waiting time increases by about 3%. Moreover, all products see a higher demand and the profits for several items increase.

In the final period `ATO` 4 the production facility experiences holding costs rising by 5%. Moreover, the profits of several items drop slightly.

When comparing the performance of the three algorithms given in Fig 4.5 and

Fig 4.6 for the task of selecting the inventory levels used by the stocking policy, we see that `wsKG` consistently performs significantly better than the other two baseline methods, achieving about 95% of the optimum after only ten samples. Between the two baseline methods, the `KG` policy achieves better solutions than `EGO` for the Assemble-To-Order problem. Looking closer, `KG`'s solution is typically about 25% away of the optimum after 10 steps, about 15% after 20 steps, and still about 4% after 50 steps, depending on the instance. `EGO` achieves less than half of the optimum after 20 iterations, and only about $50-60\%$ after 50 steps. Notably, `EGO` shows a large discrepancy between the performance of single runs; the error bar displays the mean averaged over 100 runs.

4.5 Summary

We have presented a novel Bayesian optimization algorithm that takes advantage of the additional auxiliary sources for the primary objective when they are available. In modeling those auxiliary sources, we took a rigorous treatment of the uncertainties arising from the inherent model discrepancy of the auxiliary sources, i.e. their inability to model the true objective accurately. The resulting sampling policy trades off the predicted benefit and its cost naturally.

The empirical analyses demonstrate that such treatment in modeling the primary objective and the auxiliary sources significantly improves the performance of our algorithm, and makes it outperforms the other state-of-the-art Bayesian optimization algorithms in both the MISO problems and the warm starting problems.

CHAPTER 5

CONCLUSION

In this thesis we have considered application of Bayesian optimization to the problems with parallel function evaluations, discrete function domain, and multiple information sources, which typically arise in the fields including biochemistry, aerospace engineering, machine learning, etc. We developed novel Bayesian optimization methods for these problems, and showed that the proposed methods are one-step Bayes optimal, or near-optimal with performance guarantee, and they perform well numerically in the problems considered in Chapters 2, 3, and 4. In Chapter 3, we also applied our method to a peptide optimization problem arising in biochemistry, and achieved great experimental result in finding the target peptides.

With the advances in Bayesian optimization presented here, we see the promising future of applying Bayesian optimization to a broader class of real problems. In the mean time, there are future directions in improving the current methods proposed in this thesis, and we give a short list here.

- [97] considers the same problem setting discussed in Chapter 2, that allows parallel function evaluations. Inspired by the work in Chapter 2, they extended the *knowledge gradient* to its parallel version, which is better in handling noisy function evaluations than the *expected improvement*. They demonstrated the superior performance of the *parallel knowledge gradient* method in tuning hyperparameters of practical machine learning algorithms, including logistic regression and neural networks.
- The method proposed in Chapter 3 considers a general classification problem

in biochemistry. A natural extension is to consider a regression problem with the similar experimental setting, but all existing approaches for this problem setting [94, 26, 86] quickly become too expensive to proceed as the number of parallel function evaluations grows large.

- There are occasions when the domain of the objective function is a mixture of discrete and continuous variables, and to our knowledge, there is no Bayesian optimization method proposed so far to deal with this setting. With the popularity of this problem in the engineering systems, it is an interesting topic worth pursuing.

In addition to developing the methodologies, the author, along with engineers at Yelp, also developed an open source software package for Bayesian optimization, "Metrics Optimization Engine" [12], which implements the *expected improvement* for both sequential and parallel function evaluations. The software package offers interfaces in both C++ and Python, and has been used in multiple technology firms to improve their machine learning algorithms.

The author sincerely hopes that the work in this thesis will broaden the scope of Bayesian optimization, and be valuable to application of Bayesian optimization to the problems arising in the engineering fields.

BIBLIOGRAPHY

[1] A. Abdollahzadeh, A. Reynolds, M. Christie, D. W. Corne, B. J. Davies, G. J. Williams, et al. Bayesian optimization algorithm applied to uncertainty quantification. *SPE Journal*, 17(03):865–873, 2012.

[2] S. Agarwal, D. Dugar, and S. Sengupta. Ranking chemical structures for drug discovery: a new machine learning approach. *Journal of chemical information and modeling*, 50(5):716–731, 2010.

[3] D. Allaire and K. Willcox. A mathematical and computational framework for multifidelity design and analysis with computer models. *International Journal for Uncertainty Quantification*, 4(1), 2014.

[4] X. Amatriain. 10 lessons learned from building ml systems. https://www.youtube.com/watch?v=WdzWPuazLA8, 2014. Recording of presentation from MLconf 2014, Accessed: 2015-11-26.

[5] V. Balabanov and G. Venter. Multi-fidelity optimization with high-fidelity analysis and low-fidelity gradients. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2004.

[6] P. J. Ballester and J. B. Mitchell. A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking. *Bioinformatics*, 26(9):1169–1175, 2010.

[7] E. V. Bonilla, K. M. Chai, and C. Williams. Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

[8] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[9] Y. Chen, A. Krause, and E. T. H. Zurich. Near-optimal Batch Mode Active Learning and Adaptive Submodular Optimization. 28, 2013.

[10] C. Chevalier and D. Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In *Learning and Intelligent Optimization*, pages 59–69. Springer, 2013.

[11] S. Clark. Introducing moe: Metric optimization engine; a new open source, machine learning service for optimal experiment design. http://engineeringblog.yelp.com/2014/07/introducing-moe-metric-optimization-engine-a-new-open-source-machine-learning-service-fo html, 2014. Accessed: 2015-11-26.

[12] S. C. Clark, E. Liu, P. I. Frazier, J. Wang, D. Oktay, and N. Vesdapunt. Metrics optimization engine. http://yelp.github.io/MOE/, 2014. Accessed: 2016-02-28.

[13] J. E. Dennis, Jr and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991.

[14] L. Dixon and G. Szegö. The global optimization problem: an introduction. *Towards global optimization*, 2:1–15, 1978.

[15] M. S. Eldred, A. A. Giunta, and S. S. Collis. Second-order corrections for surrogate-based optimization with model hierarchies. In *Proceedings of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pages 2013–2014, 2004.

[16] M. Feurer, J. T. Springenberg, and F. Hutter. Using meta-learning to initialize bayesian optimization of hyperparameters. In *Proceedings of the International Workshop on Meta-Learning and Algorithm Selection (ECAI)*, pages 3–10, 2014.

[17] E. L. First, M. Hasan, and C. A. Floudas. Discovery of novel zeolites for natural gas purification through combined material screening and process optimization. *AIChE Journal*, 60(5):1767–1785, 2014.

[18] A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[19] P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

[20] P. Frazier, W. Powell, and S. Dayanik. The knowledge gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613, 2009.

[21] P. I. Frazier and J. Wang. Bayesian optimization for materials design.

In *Information Science for Materials Discovery and Design*, pages 45–75. Springer, 2016.

[22] P. I. Frazier, J. Xie, and S. E. Chick. Value of information methods for pairwise sampling with correlations. In *Proceedings of the Winter Simulation Conference*, pages 3979–3991. Winter Simulation Conference, 2011.

[23] A. Genz. Numerical computation of multivariate normal probabilities. *Journal of computational and graphical statistics*, 1(2):141–149, 1992.

[24] H. M. Geysen and T. J. Mason. Screening chemically synthesized peptide libraries for biologically-relevant molecules. *Bioorganic & Medicinal Chemistry Letters*, 3(3):397–404, 1993.

[25] D. Ginsbourger. Two advances in gaussian process-based prediction and optimization for computer experiments. In *MASCOT09 Meeting*, 2009.

[26] D. Ginsbourger, R. Le Riche, and L. Carraro. A multi-points criterion for deterministic parallel global optimization based on kriging. In *NCP07*, 2007.

[27] D. Ginsbourger, V. Picheny, O. Roustant, et al. Diceoptim: Kriging-based optimization for computer experiments. https://cran.r-project.org/web/packages/DiceOptim/index.html, 2015. Accessed: 2016-02-13.

[28] P. Glasserman. *Gradient estimation via perturbation analysis*. Springer Science & Business Media, 1991.

[29] P. Goovaerts. *Geostatistics for Natural Resources Evaluation*. Oxford University, 1997.

[30] L. L. Gratiet and C. Cannamela. Cokriging-based sequential design strategies using fast cross-validation techniques for multi-fidelity computer codes. *Technometrics*, 57(3):418–427, 2015.

[31] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research*, 13(1):1809–1837, 2012.

[32] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems*, pages 918–926, 2014.

[33] Y.-C. Ho. Performance evaluation and perturbation analysis of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 32(7):563–572, Jul 1987.

[34] S. C. H. Hoi, R. Jin, and M. R. Lyu. Large-scale text categorization by batch mode active learning. *Proceedings of the 15th international conference on World Wide Web - WWW '06*, page 633, 2006.

[35] S. C. H. Hoi, R. Jin, J. Zhu, and M. R. Lyu. Batch mode active learning and its application to medical image classification. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 417–424, 2006.

[36] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[37] L. J. Hong and B. L. Nelson. Discrete optimization via simulation using compass. *Operations Research*, 54(1):115–129, 2006.

[38] D. Huang, T. Allen, W. Notz, and N. Zeng. Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models. *Journal of Global Optimization*, 34(3):441–466, 2006.

[39] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization*, 34(3):441–466, 2006.

[40] A. I. J. Forrester, A. J. Keane, and N. W. Bressloff. Design and analysis of" noisy" computer experiments. *AIAA journal*, 44(10):2331–2339, 2006.

[41] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

[42] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[43] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online; accessed 2014-12-01].

[44] K. Kandasamy, G. Dasarathy, J. B. Oliva, J. Schneider, and B. Poczos. Multi-fidelity gaussian process bandit optimisation. *arXiv:1603.06288*, 2016.

[45] A. Z. Kattamis, R. J. Holmes, I.-C. Cheng, K. Long, J. C. Sturm, S. R. Forrest, and S. Wagner. High mobility nanocrystalline silicon transistors on clear plastic substrates. *IEEE electron device letters*, 27(1):49–51, 2006.

[46] J. Kennedy. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 760–766. Springer, 2010.

[47] M. C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[48] S.-H. Kim and B. L. Nelson. Recent advances in ranking and selection. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, pages 162–172. IEEE Press, 2007.

[49] D. B. Kitchen, H. Decornez, J. R. Furr, and J. Bajorath. Docking and scoring in virtual screening for drug discovery: methods and applications. *Nature reviews Drug discovery*, 3(11):935–949, 2004.

[50] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

[51] N. M. Kosa, R. W. Haushalter, A. R. Smith, and M. D. Burkart. Reversible labeling of native and fusion-protein motifs. *Nature Methods*, 9(10):981–984, 2012.

[52] H. Kushner and G. G. Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

[53] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):97–106, 1964.

[54] R. Lam, D. Allaire, and K. Willcox. Multifidelity optimization using statistical surrogate modeling for non-hierarchical information sources. In *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2015.

[55] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Last Accessed on 10/09/2016.

[56] Y. Li, H. Liu, and W. Powell. A lasso-based sparse knowledge gradient policy for sequential optimal learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 417–425, 2016.

[57] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[58] D. J. Lizotte. *Practical bayesian optimization*. University of Alberta, 2008.

[59] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*, chapter Text classification and Naive Bayes. Cambridge University Press, New York, NY, USA, 2008.

[60] A. March and K. Willcox. Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives. *AIAA Journal*, 50(5):1079–1089, 2012.

[61] S. Marmin, C. Chevalier, and D. Ginsbourger. Differentiating the multipoint expected improvement for optimal batch design. In *Machine Learning, Optimization, and Big Data*, pages 37–48. Springer, 2015.

[62] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[63] J. Mockus. *The bayesian approach to local optimization*. Springer, 1989.

[64] J. Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

[65] J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[66] D. M. Negoescu, P. I. Frazier, and W. B. Powell. The knowledge-gradient algorithm for sequencing experiments in drug discovery. *INFORMS Journal on Computing*, 23(3):346–363, 2011.

[67] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations

for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[68] V. Picheny, D. Ginsbourger, Y. Richet, and G. Caplin. Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics*, 55(1):2–13, 2013.

[69] M. Poloczek, J. Wang, and P. I. Frazier. Multi-information source optimization with general model discrepancies. *ArXiv e-print 1603.00389*, 2016. Available via http://arxiv.org/abs/1603.00389.

[70] M. Poloczek, J. Wang, and P. I. Frazier. Warm starting bayesian optimization. *arXiv preprint arXiv:1608.03585*, 2016.

[71] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

[72] H. Rabitz and K. Shim. Multicomponent semiconductor material discovery guided by a generalized correlated function expansion. *The Journal of Chemical Physics*, 111(23):10640–10651, 1999.

[73] D. Rajnarayan, A. Haas, and I. Kroo. A multifidelity gradient-free optimization method and application to aerodynamic design. In *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, number 6020, 2008.

[74] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[75] H. L. Royden and P. Fitzpatrick. *Real analysis*, volume 198. Macmillan New York, 1988.

[76] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

[77] T. J. Santner, B. J. Williams, and W. I. Notz. *The design and analysis of computer experiments*. Springer Science & Business Media, 2013.

[78] M. J. Sasena. *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*. PhD thesis, General Motors, 2002.

[79] W. Scott, P. Frazier, and W. Powell. The correlated knowledge gradient for simulation optimization of continuous parameters using gaussian process regression. *SIAM Journal on Optimization*, 21(3):996–1026, 2011.

[80] B. K. Shoichet. Virtual screening of chemical libraries. *Nature*, 432(7019):862–865, 2004.

[81] B. C. Smith, B. Settles, W. C. Hallows, M. W. Craven, and J. M. Denu. Sirt3 substrate specificity determined by peptide arrays and machine learning. *ACS chemical biology*, 6(2):146–157, 2010.

[82] G. P. Smith and V. A. Petrenko. Phage display. *Chemical reviews*, 97(2):391–410, 1997.

[83] S. P. Smith. Differentiation of the cholesky algorithm. *Journal of Computational and Graphical Statistics*, 4(2):134–147, 1995.

[84] J. Snoek. Personal communication, 2016.

[85] J. Snoek and et al. Spearmint. http://github.com/HIPS/Spearmint. Last accessed on 05/18/2016.

[86] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[87] K. Swersky, J. Snoek, and R. P. Adams. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012, 2013.

[88] Y.-W. Teh, M. Seeger, and M. Jordan. Semiparametric latent factor models. In *Artificial Intelligence and Statistics 10*, 2005.

[89] Theano. Theano: Logistic regression. http://deeplearning.net/tutorial/code/logistic_sgd.py. Last Accessed on 10/08/16.

[90] A. Torn and A. Zilinskas. *Global optimization*. Springer-Verlag New York, Inc., 1989.

[91] USPS. USPS dataset. http://mldata.org/repository/data/viewslug/usps/. Last Accessed on 10/09/2016.

[92] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

[93] J. Villemonteix, E. Vazquez, and E. Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

[94] J. Wang, S. C. Clark, E. Liu, and P. I. Frazier. Parallel bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*, 2016.

[95] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[96] R. L. Winkler. Combining probability distributions from dependent information sources. *Management Science*, 27(4):479–488, 1981.

[97] J. Wu and P. I. Frazier. The parallel knowledge gradient method for batch bayesian optimization. *arXiv preprint arXiv:1606.04414*, 2016.

[98] J. Xie, P. I. Frazier, and S. Chick. Assemble to order simulator. http://simopt.org/wiki/index.php?title=Assemble_to_Order&oldid=447, 2012. Accessed May 9, 2016.

[99] J. Xie, P. I. Frazier, and S. E. Chick. Bayesian optimization via simulation with pairwise sampling and correlated prior beliefs. *Operations Research*, 2013. to appear.

[100] J. Yin, P. D. Straight, S. M. McLoughlin, Z. Zhou, A. J. Lin, D. E. Golan, N. L. Kelleher, R. Kolter, and C. T. Walsh. Genetically encoded short peptide tag for versatile protein labeling by sfp phosphopantetheinyl transferase. *Proceedings of the National Academy of Sciences of the United States of America*, 102(44):15815–15820, 2005.

[101] Y. Zhang, W. Xu, and J. Callan. Exploration and exploitation in adaptive filtering based on bayesian active learning. In *ICML*, volume 3, pages 896–904, 2003.

[102] Z. Zhou, P. Cironi, A. J. Lin, Y. Xu, S. Hrvatin, D. E. Golan, P. A. Silver, C. T. Walsh, and J. Yin. Genetically encoded short peptide tags for orthogonal protein labeling by sfp and acps phosphopantetheinyl transferases. *ACS Chemical Biology*, 2(5):337–346, 2007.